An Example-Based Transfer System

Hideo Watanabe

April, 1996

An Example-Based Transfer System

Hideo Watanabe

April, 1996

Abstract

Automatic translation methodologies have been extensively studied for decades at many academic research institutions around the world, and many companies have also been developed machine translation (MT) systems as well. Some of them can be considered successful, but they are not sufficiently advanced to be used instead of professional translators. There is a large quality gap between outputs of human translators and those of MT systems. It is extremely difficult for MT systems to match the quality of human translators, because they must simulate many processes in the brain, such as common-sense reasoning using knowledge of the world, and the mechanism of the brain has not yet been fully explored even in the field of medical science. Nonetheless, there are various lines of research on improving MT systems. One of them is the example-based approach, which uses translation examples directly on the basis of the similarity, just as a person uses memories of past translations when trying to translate a new sentence.

This thesis describes an example-based transfer system and proposes some methodologies related to example-based transfer. As a fundamental framework for a transducing mechanism, the *pattern combination transfer* or (PCT) model is proposed. The PCT model produces an output structure by non-destructively combining target parts of translation patterns, each of which consists of dependency structures in source and target languages and correspondences between them. This proposed mechanism employs more extended mapping of correspondences than the one-to-one mapping used in conventional transfer systems, to allow expression of some peculiar or exceptional translation phenomena.

In addition to the PCT model, the construction of an example-based transfer system requires a method for calculating the similarity between an input and translation patterns, and a method for selecting appropriate translation patterns for the input.

To meet the first requirement, a method is proposed for calculating the similarity of Japanese dependency structures. This method can deal uniformly with translation patterns derived from translation examples and those derived from conventional transfer knowledge, even if they can be expressed as dependency structures. This means that conventional transfer knowledge can be used as a fail-safe mechanism if appropriate examples are not found. To meet the second requirement, an efficient method is proposed for finding a set of translation patterns that are appropriate from the viewpoint of their overall similarity.

Making a translation pattern from a translation example is not simple, because it requires the parsing process to create dependency structures in the source and target languages, and has to determine correspondences between these structures. The cost of this task must be reduced. I propose a method for creating translation patterns by comparing a MT result and its correct translation. This is not a completely automatic system, but it can greatly reduce the cost of creating such translation patterns.

Since it is an open question how many translation patterns are sufficient for a particular domain of translation, we must consider how to obtain a translation even when the number of translation patterns is not sufficient. One solution is to use translation patterns derived from conventional transfer knowledge as a fail-safe mechanism. But, this creates another problem called *example interference* in which side effects are caused by the selection of inappropriate translation patterns. To avoid such problems, I propose a method for identifying exceptional translation patterns that may cause side effects.

Acknowledgments

I would like to express my sincere appreciation to Professor Makoto Nagao of Kyoto University for supervising this thesis and for his continuous encouragement and guidance.

I would also like to thank Professor Satoshi Sato of the Japan Advanced Institute of Science and Technology, Hokuriku, and Dr. Hiroshi Maruyama of IBM Tokyo Research Laboratory for their constructive and fruitful discussions on this research topic. Particularly, the work on the efficient cover search algorithm described in this thesis was done with Dr. Hiroshi Maruyama.

I am grateful to my colleagues, Koichi Takeda, Hiroshi Nomiyama, Shiho Ogino, Tetsuya Nasukawa, and Naohiko Uramoto, in the Natural Language Processing group at IBM Tokyo Research Laboratory, especially Dr. Taijiro Tsutsumi and Mr. Masayuki Morohashi, for their management support in this research.

Finally, I would like to thank Mr. Michael McDonald for his intensive and patient proof-reading of the manuscript of this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Difficulties in the Transfer Phase of MT

The transfer process in machine translation systems is, in general, more complicated than the processes of analysis and generation. It must usually handle not only lexical selection but also structural changes at the same time. A typical conventional transfer system is configured as a tree-to-tree rewriting system, in which many mutually dependent[1] rules are executed. This kind of system involves the problem that a grammar writer cannot grasp the behavior of all the rules for a very wide variety of input situations, with the result that a new rule may have unforeseen side effects. The fundamental causes of this problem are that rules are mutually dependent and that they rely heavily on human linguistic knowledge, or the linguistic intuition of the rule writers, because no linguistic theory exists for lexical transfer [Melby 86] and the transfer task is inherently a conglomeration of individual lexical rules [Nitta 86] [Tsujii and Fujita 90]. This implies that the transfer process falls into a class of problem that cannot easily be controlled by the linguistic intuition of rule writers.

To overcome this transfer bottleneck, rules should be

- designed to be as mutually independent as possible, and

[1] This means that the calling sequence of rules constructs a complicated, huge decision network, in which some rules are designed to be called before or after specific rules. In other words, most such rules cannot perform a particular function alone.

- selected according to a criterion.

In accordance with these observations, various attempts have been made to overcome the problems of transfer: they include the knowledge-based or interlingual approach [MTJ 89a] [MTJ 89b] [Tomita and Carbonell 86], which arrives at an output via an interlingual representation of an input sentence; bilingual signs [Tsujii and Fujita 90], which tries to formulate the transfer phase as logical inference; Synchronous Tree-Adjoining Grammar [Abeillé et al. 90], which expresses a transfer rule as a pair of derivational trees in both languages and produces an output by synchronous derivation when the parsing is finished; and the Shake-and-Bake approach [Whitelock 92], in which a powerful generation module creates an output from a set of lexical items in the target language derived from an input sentence. An emerging technology, the *example-based approach*[2] [Nagao 84] [Salder 89] [Sato and Nagao 90] [Sumita et al. 90] [Furuse and Iida 92], is also suitable for constructing such a system. The essential idea of this example-based approach is that the system chooses from a database of examples some examples similar to the given input, and applies to the input the knowledge attached to the chosen examples. In the next section, I describe the framework of example-based translation more precisely.

1.2 Framework of an Example-Based Transfer System

Supposing that there is a corpus of parsed translation examples in which corresponding parts are linked to each other, we can regard those parsed translation examples as translation rules (or translation patterns) in the transfer phase. A promising approach is therefore to make a transfer process that (1) chooses a set of translation examples, in each of which a source part is similar to a part of the input, and source parts of which overlap the input, and (2) constructs an output by combining the target parts of the chosen translation examples.

In Figure 1.1, for example, (a) is the parsed dependency structure of an input Japanese sentence, "Kare ga kusuri wo nomu." Suppose that (b) is selected from the translation pattern-base as the most similar translation pattern for the part

[2]This approach can be regarded as an application of case-based reasoning [Kolodner and Riesbeck 89] to machine translation.

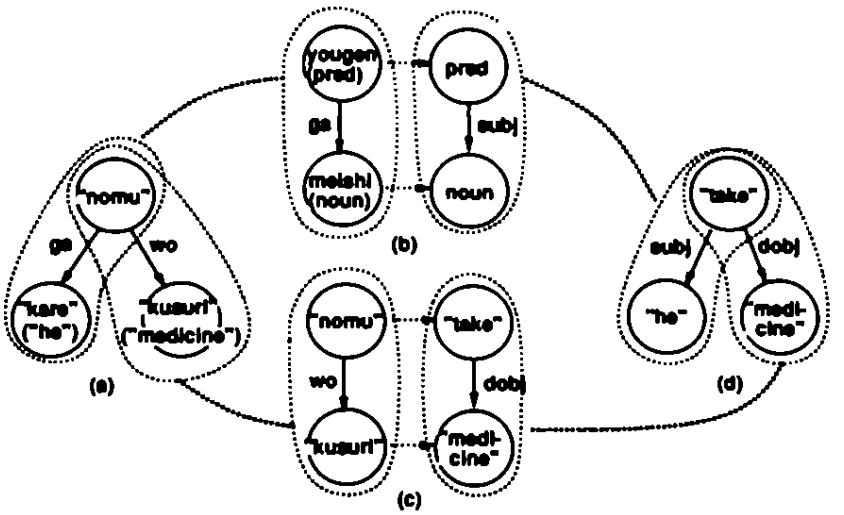

Figure 1.1: Sample Japanese-to-English translation

"kare ga ... nomu," and that (c) is selected as the most similar translation pattern for the part "kusuri wo nomu," even though there are several translation candidates for the Japanese verb "nomu." This figure illustrates what we would like to do; that is, to construct (d), the translated structure, by combining the target structures of the selected translation patterns.

Figure 1.2 shows the framework of an example-based transfer system, in which the *matching* part calculates the similarity between an input structure and translation patterns, the *pattern selection* part selects the most appropriate set of translation patterns for the input, and the *construction* part deals with the transducing task; that is, it produces an output structure from the selected translation patterns. To develop this kind of system, we must establish

(1) a measure for similarity,

(2) a mechanism for selecting the most appropriate set of translation patterns,

(3) a mechanism for combining target parts of translation patterns, and

(4) correspondence between the source part and the target part of a translation pattern.

Item (1) corresponds to the matching part, (2) to the pattern selection part, and (3)

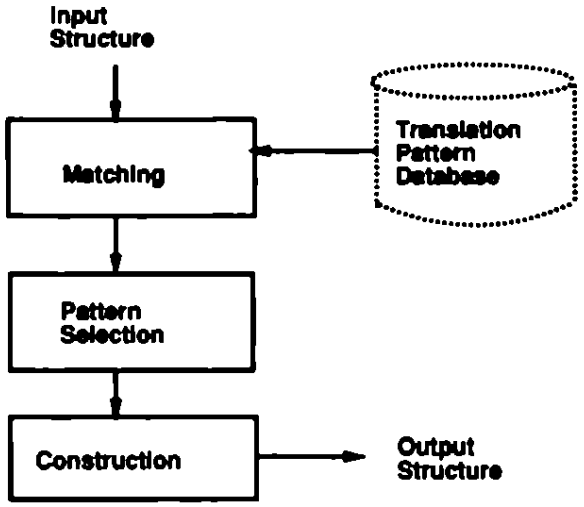

Figure 1.2: Framework of Example-based Transfer System

and (4) to the construction part. We have developed an example-based transfer system, called *similarity-driven transfer system* or *SimTran*. For item (1), *SimTran* includes a similarity calculation method for Japanese dependency structures; for item (2), an efficient method of searching for an appropriate set of translation patterns [Maruyama and Watanabe 92]; and for item (3) and (4), a model called *Pattern Combination Transfer (PCT)*. A detailed description of these elements of *SimTran* will be given in subsequent chapters.

1.3 Creating and Maintaining Translation Patterns

Although the adoption of the example-based approach simplifies the transfer phase, some problems still remain. One is the high cost of creating translation patterns. The reason of this is that a translation pattern is configured as a pair of dependency structures in the source and target languages and requires correspondences among nodes of these two dependency structures.[3] To create such a translation pattern,

[3]This problem disappears if the example-based translation system uses only source-level translation pairs, and thus does not require any parsing. Unfortunately, such a system is almost impossible

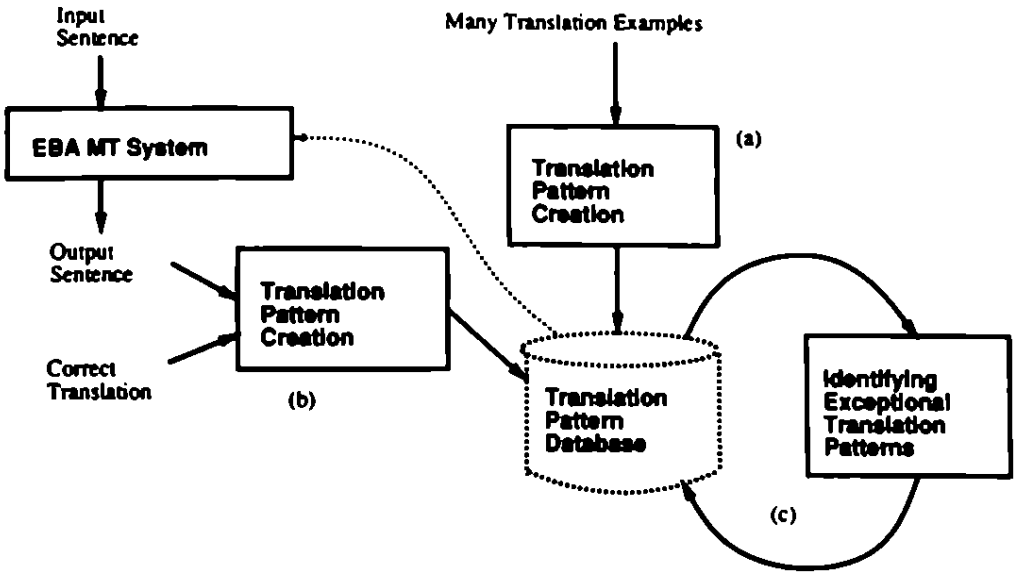

Figure 1.3: Framework of Creating and Maintaining Translation Patterns

we need the help of the parsing modules for the two languages and a method for finding correspondences.

It is thus important to reduce the cost of creating translation patterns. There are two approaches to creating translation patterns automatically or semi-automatically: one is to create translation patterns from a large number of translation examples ((a) in Figure 1.3); the other is to create necessary translation patterns from wrong translations and their corrections ((b) in Figure 1.3). The first must be used to make a translation pattern database from scratch, while the second is suitable for enhancing the current translation pattern database in a boot-strapping manner. The latter method will be described in Chapter 4.

Further, there is another problem, that is, how many examples should be prepared. This depends on the target domain of the translation, and is probably an open question. An example-based translation system must be usable even in situations where insufficient examples or translation patterns are provided. One approach to meeting this requirement is to use translation patterns derived from conventional transfer knowledge, which are mostly syntax-level descriptions. For example, the Japanese pattern "NP ga V" is translated as the English pattern "NP(+subj) V,"

to design at the current stage of processes in natural language processing technologies.

as a fail-safe mechanism if no similar translation pattern is found. This issue will be dealt with in Chapter 3.

A different type of problem called *example interference* is caused by a lack of sufficient examples; in this case, selection of an inappropriate translation pattern has side effects on an output structure. Since the example-based MT system selects translation patterns simply according to similarity, this problem tends to occur particularly when there are not enough examples for a particular domain of translation. To avoid this problem, it is necessary to identify translation patterns that might have side effects, that is, to find exceptional translation patterns ((c) in Figure 1.3). This issue will be dealt with in Chapter 5.

1.4 Outline of the Thesis

Chapter 2 proposes a basic model for the transfer phase, called Pattern Combination Transfer (PCT), which is suitable for example-based machine translation. PCT includes a transducing mechanism and a definition of translation patterns.

Chapter 3 describes our example-based transfer system, called *SimTran*, as an instance of a system using PCT. It also gives a measure for determining the similarity between two Japanese dependency structures, and a method for selecting the most appropriate set of translation patterns.

Chapter 4 deals with how to make translation patterns, and describes a method for constructing a new translation pattern from an MT result and its correction.

Chapter 5 deals with a problem called *example interference*, and describes a way of identifying exceptional translation patterns in a translation pattern database.

Chapter 6 offers some concluding remarks.

Chapter 2

Pattern Combination Transfer (PCT)

2.1 Introduction

This chapter describes a basic transfer model called *Pattern Combination Transfer (PCT)*,[1], which is suitable for example-based transfer systems.

Basically, PCT is a transducing mechanism that, when supplied with a set of translation patterns each of which is a pair of dependency structures in the source and target languages, produces an output dependency structure by combining the target parts of the given translation patterns. For instance, Figure 2.1 shows an example of translation using this mechanism. Here, (a) is an input dependency structure, (b) and (c) are selected translation patterns, and (d) is the output structure constructed by combining the target parts of (b) and (c).

PCT also includes a representational framework for translation patterns. Translation patterns in the PCT model have a more extended correspondence between the source and target parts than simple one-to-one mapping. This makes it easier to create a concise translation pattern.

[1]This was previously called *Rule Combination Transfer (RCT)*, but I have renamed it *Pattern Combination Transfer* in this thesis to avoid potential confusion resulting from the use of the word *rule*.

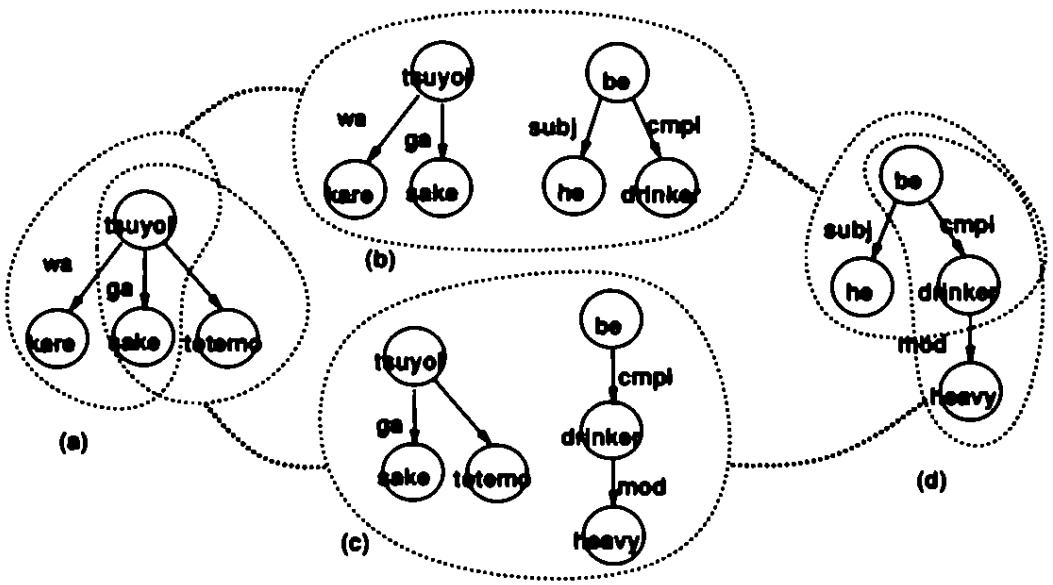

Figure 2.1: An example of example-based translation

The proposed model, PCT, is intended to be used as a foundation of example-based transfer systems, but it does not assume that all translation patterns should be extracted from examples; rather, it can accept any kind of translation pattern that is expressed as a pair of dependency structures in the source and target languages. Therefore, PCT can be used for any kind of transducing task, including example-based MT.

The next section defines the basic data structure used in this model and the format of a translation pattern. Section 2.3 describes the conditions that must be satisfied by the selected translation patterns. Section 2.4 describes the mechanism for matching a dependency structure of an input sentence with the source parts of translation patterns. Section 2.5 introduces a method for constructing a target structure from the selected translation patterns. Section 2.6 gives examples of translation according to the proposed model. Section 2.7 discusses related work and several issues. The chapter ends with some concluding remarks.

2.2 Data Structure

This section describes data structures and introduces several terms used in subsequent sections.

The basic data structure used in this thesis is a labeled directed graph, or **LDG**, which has directed, labeled arcs and labeled nodes. We call an LDG that has only one root a *rooted* labeled directed graph (**RLDG**). Further, we call an RLDG whose arcs are acyclic a rooted labeled directed *acyclic* graph (**RLDAG**).[2]

An LDG G is defined, following Ehrig [Ehrig 79], as

$$G = \langle N, A, s, t, fn, fa \rangle$$

where N is a set of nodes, A is a set of arcs, s is a function giving the source node of an arc, t is a function giving the target node of an arc, fn is a function giving a set of labels of a node (or node label-set), and fa is a function giving a label of an arc.[3]

An input dependency structure is assumed to be an RLDAG whose node label-sets are unique; that is, for any two distinct nodes, there is no common label in these label-sets.

2.3 Translation Pattern

A translation pattern pt consists of the following three components:

$$pt = \langle G_m, G_c, M \uparrow\downarrow \rangle$$

where G_m is a matching graph, G_c is a construction graph, and $M \uparrow\downarrow$ is a structural mapping from G_m to G_c. The matching graph G_m and the construction graph G_c

[2]The term **DAG** is often used in the NLP world, and usually denotes a rooted connected labeled (as functional) directed graph. In this thesis, however, **DAG** denotes a directed acyclic graph that may have multiple roots, is not necessarily a connected graph, and does not necessarily have labels. Therefore, a feature structure of the type used in LFG [Kaplan and Bresnan 82], and other grammars is called an RCLFDAG in this thesis.

[3]Unlike in Ehrig's model, the label function fn is extended to return a set of labels for a node, because nodes need to have more than one label in the model described in this section.

must be RLDAGs.[4]

A **structural mapping** designates the correspondences from a subset of G_m nodes to a subset of G_e nodes. For instance, in Figure 2.2, the Japanese word "nagai" ("long") should correspond to both of the English words "have" and "long," because if another word governs "nagai" then its English translation should be connected to the word "have," and if the Japanese word "itsumo" ("always") modifies "nagai" then its translation "always" must modify "have." On the other hand, if the Japanese word "totemo" ("very") modifies "nagai" then its translation "very" should be connected to "long." This shows that for a node in a source language, two kind of connection points, for translations of both governing structures and governed structures of the node, are needed in its translation structure.[5] We call a mapping from a G_m node to a upper connection node in G_e an **upward mapping**, and a mapping from a G_m node to a lower connection node in G_e a **downward mapping**, and denote these two kinds of mapping as follows:

$$M \uparrow\downarrow = (M \uparrow, M \downarrow)$$

where $M \uparrow$ is upward mapping, and $M \downarrow$ is downward mapping.

Not all kinds of mapping should be permitted as $M \uparrow$ and $M \downarrow$. A translation pattern $pt = \langle G_m, G_e, M \uparrow\downarrow \rangle$ must satisfy the following conditions:

(1) $M \uparrow$ and $M \downarrow$ are bijections between a subset of G_m and a subset of G_e, and

(2) $M \uparrow(root(G_m)) = root(G_e)$.

Condition (1) ensures that there is only one connection point in G_e of a G_m node for upward and downward mappings, respectively, and condition (2) ensures that the target part of the structure governing the G_m root is connected to the root of G_e.

This translation pattern can be used bi-directionally. If the translation direction is

[4]Such graphs are sufficient to express almost all linguistic structures.

[5]We can consider the more complicated mapping such that a source node has more than two corresponding nodes in the target structure. The more complicated the correspondences are, the less efficient the system is. In other words, the expressive power of a translation pattern and the efficiency of the system are the relation of trade-off. Therefore, PCT employs the upward and downward mappings.

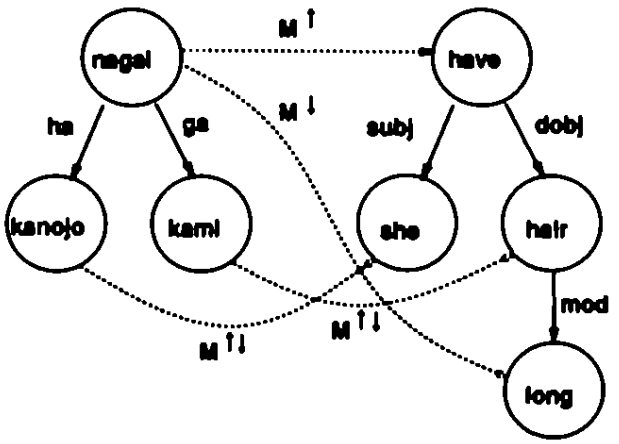

Figure 2.2: A sample translation pattern for Japanese and English

from G_e to G_m, then we can use a translation pattern by swapping both G_m and G_e, and also the domains and ranges of the mappings. In what follows, we assume that the translation direction is from G_m to G_e.

2.4 Matching

Given an input G_{in}, we must select a set of translation patterns whose source parts completely cover the input.[6] There are several methods for selecting the most appropriate set of translation patterns.[7] Therefore, in this section, we will not mention a specific method, but will rather provide several conditions that the selected set of translation patterns must satisfy.

First of all, we will define matching between the source part of a translation pattern and an input. An LDG $G_1(=(N_1,A_1,s_1,t_1,fn_1,fa_1))$ **matches** $G_2(=(N_2,A_2,s_2,t_2,fn_2,fa_2))$ if and only if there is a mapping Ψ, from an LDG G_1 to an LDG G_2, that satisfies

[6] In real applications, there would be cases in which several parts of the input structure are not covered by appropriate translation patterns. To cover such cases, what are called catch-all translation patterns should be supplied. Chapter 3 will describe our approach which provides syntax-level translation patterns that cover all kind of syntactic combinations.

[7] For instance, top-down or bottom-up searches can be easily considered as methods for finding a set of translation patterns. Chapter 3 will describe an efficient search mechanism that can select the most appropriate pattern set from several candidates.

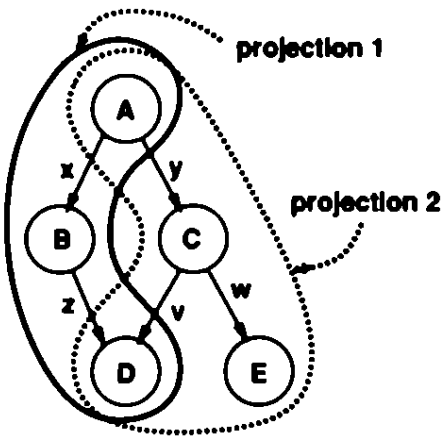

Figure 2.3: An isomorphic cover

the following conditions:

(1) Ψ is a bijection consisting of $\Psi_n : N_1 \rightarrow N_2$ and $\Psi_a : A_1 \rightarrow A_2$, and

(2) For each x in A_1, $fa_1(x) = fa_2(\Psi_a(x))$, $s_1(x) = \Psi_n^{-1}(s_2(\Psi_a(x)))$, and $t_1(x) = \Psi_n^{-1}(t_2(\Psi_a(x)))$.

If an LDG G_1 matches an LDG G_2, then they are also said to be **isomorphic**. If an LDG G_1 matches a sub-LDG G_2' on an LDG G_2, then the sub-LDG G_2' is called a **projection** of G_1. If there is a projection pj of G_1 on G_2, then G_1 is called an **origin graph** of the projection pj and, for each node and arc in the projection pj, their corresponding nodes and arcs in G_1 are also called **origin nodes** and **arcs**, respectively.

A set of projections on an LDG G is called a **cover** of G. What we need here is a cover whose projections completely overlap an input LDG G_{in}, that is, a cover such that the union of its projections is isomorphic to the input. We call such a cover an **i-cover**, for simplicity. In Figure 2.3, for instance, the cover consisting of projections 1 and 2 is an i-cover.

Therefore, G_m's of selected translation patterns must construct an i-cover on an input G_{in}.

2.5 Transducing

Given a set of translation patterns whose projections make an i-cover on G_{in}, we must construct a target structure by combining the target parts of the selected patterns. There may, however, be cases in which one translation pattern has more than one projection on G_{in}. Therefore, before a target structure is constructed, a copy of the original pattern must be produced for each projection. We call a copy of a translation pattern a **pattern instance**.

Basically, a target structure is constructed by combining the target parts of selected pattern instances. To determine which nodes are to be merged into a single node, node label-sets are used.

2.5.1 Node Labeling

We assume that each node in G_{in} has a unique label-set, but the node label-sets of G_m and G_c of translation patterns (precisely pattern instances) have not been mentioned. This section describes how to determine the label-sets of G_m and G_c.

The label-sets of G_m and G_c nodes in given pattern instances are determined as follows:

G_m Node Labeling: Each node in G_m is given the same label-set as its matching node in G_{in}.

G_c Node Labeling: The label-set of a node n_c in G_c is a union of label-sets of nodes (n_m) in G_m such that $n_c = M\uparrow(n_m)$ or $n_c = M\downarrow(n_m)$.

Since we adopt upward mapping and downward mapping as correspondence between G_m and G_c, there may be two distinct nodes of G_c in a pattern instance that are mapped by a node in G_m with $M\uparrow$ and $M\downarrow$, respectively. In this case, the transferred labels of these two nodes are the same, but they should be different, because these two mappings are used to designate two different nodes in a target structure. We must therefore relabel all G_c's of pattern instances as follows:

G_c Node Relabeling: Find a label l such that a G_m node having the label l relates to two distinct nodes in a G_c, and make a set Y of G_c nodes that have the

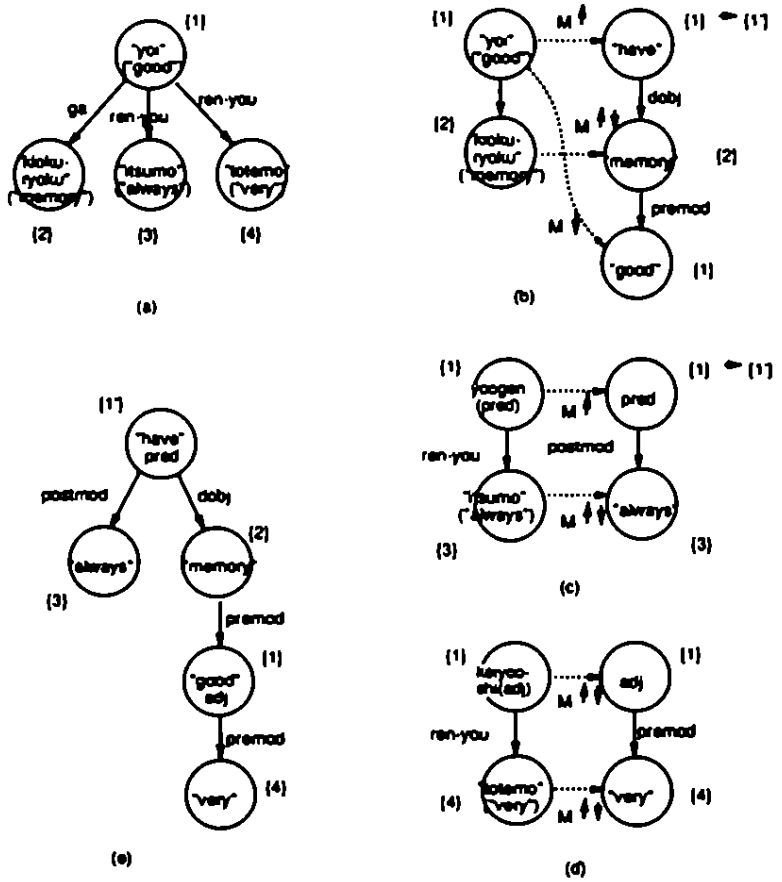

Figure 2.4: An example of node relabeling

label l from all pattern instances. For each node y in Y, if y is related only by $M\uparrow$, or is related by both $M\uparrow$ and $M\downarrow$ and has no descendant nodes, then the label l of y is changed to l'.

In Figure 2.4, (a) is an input dependency structure, (b), (c), and (d) are pattern instances, and (e) is an output dependency structure produced from the target parts of these pattern instances. In pattern instance (b), the label 1 of "yoi" is distributed to "have" and "good." In G_t's, the "have" of (b) and *pred* of (c) match the above relabeling conditions, and their labels are changed from 1 to 1'.

2.5.2 Gluing

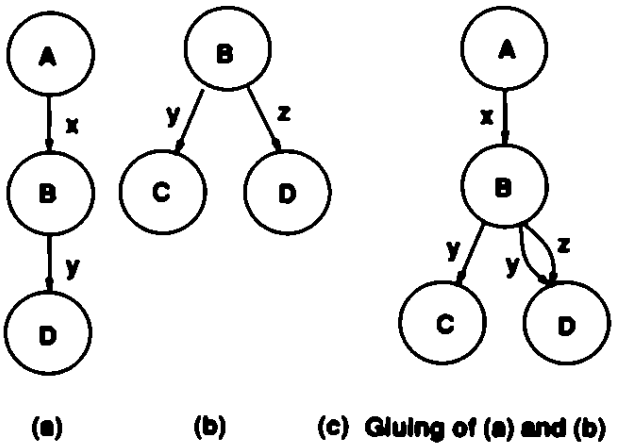

Figure 2.5: An example of gluing

Here, we introduce *gluing* as a mechanism for connecting the target parts of translation patterns.

Unification is a well-known computational method for connecting graphs, and is widely used in natural language processing. Usually, unification uses two functional RLDAGs as data and unifies them from the root node down to the leaves [Knight 89]. In the model proposed in this section, we want to merge nodes having the same label in several graphs, even if those graphs have different root nodes and are not functional, as shown in Figure 2.5. Unification, however, cannot proceed in this manner, because it unifies two nodes that occupy the same position, and always starts from the root node. For instance, in Figure 2.5 (where A, B, C, and D are labels), unification fails even if it starts from node B, since it tries to unify node D of (a) and node C of (b) for arc y.

In Graph Grammars, this method of connecting two graphs is called **gluing** [Ehrig 79].[8] Gluing is a process that, given two graph morphisms (roughly mappings) $b : K \rightarrow B$ and $d : K \rightarrow D$, where K, B, and D are LDGs, produces an LDG in which nodes with the same colors (or labels) mapped by b and d are glued.

The gluing used in Graph Grammars is not concerned with the contents of a node, and assumes that a node has a single label. It must therefore be extended in order to check the consistency among nodes to be glued, and to deal with label-sets. This

[8] A similar operation called **join** has been proposed [Sowa 84].

consistency checking, however, depends on the data structure of a node, and it is beyond the scope of this section to supply a definition of such a data structure and the consistency-checking mechanism. Therefore, we simply state here that gluing succeeds only if the nodes to be glued are consistent.[9]

Figure 2.6 shows the procedure for gluing. This procedure has two parts: one for merging nodes (line 1-6), and one for merging arcs (line 7-19). In the first part, the function *getGluingNodeSet* collects nodes to be glued in $N_g[]$ and their label-sets in $L_n[]$. We call this node set a **gluing node set**. After that, for each gluing node set, nodes are merged if they are consistent (lines 2-5). In the second part, all arcs are reattached to the original nodes, which may or may not be merged, and some arcs that have the same label and that are attached to the same source and target nodes are merged.

Let us consider the complexity of this procedure. Let k be the number of given RLDAGs, let n be $max(|N_1|, ..., |N_k|)$, and let a be $max(|A_1|, ..., |A_k|)$. Line 23 iterates at most kn times, because the maximal value of N is kn. Since the maximal value of *lnum* is also kn, line 25 iterates at most kn times. Therefore, the complexity of the function *getGluingNodeSet* is $O(k^2n^2)$. Further, line 10 iterates at most ka times, and the *idxs* (line 11) and *idxt* (line 12) searches at most k^2n^2 iterations each. Therefore, the complexity of the arc-merging part is $O(k^3n^2a)$. Hence, the total complexity of this procedure is $O(k^3n^2a)$.

As stated before, a target structure is constructed by gluing the G_c's of given pattern instances so that nodes having the same label are merged into a single node. In Figure 2.4, (e) is the result of gluing the G_c's of pattern instances (b), (c), and (d).[10] Of course, it may be necessary to check the linguistic consistency of nodes to be glued and of the glued graph. The details of such checking mechanisms, however, are beyond the scope of this section.

Since the result graph of the above sequence is not necessarily a rooted acyclic graph, several conditions for ensuring that an RLDAG is produced as a result of gluing are described in the appendix. Therefore, PCT can be an RLDAG-to-RLDAG transducer if the given pattern instances satisfy some conditions.

[9] Unification, the ψ-term [Ait-Kaci 86], or some such mechanism might be suitable for node consistency checking.

[10] For convenience, some nodes are lexicalized.

```
procedure gluing(G_1 = {N_1, A_1},...,G_k = {N_k, A_k}) begin
    (lnum, L_n[•], N_g[•]) ← getGluingNodeSet(N_1, ..., N_k)                    (1)
    for i = 1 to lnum begin                                                     (2)
        mergedNode[i] ← mergeNodesIfConsistent(N_g[i])                          (3)
        if mergedNode[i] = φ then fail                                          (4)
    end                                                                         (5)
    N ← ∪_{i=1}^{lnum} mergedNode[i]                                            (6)
    A ← ∪_{i=1}^{k} A_i                                                         (7)
    mergedArc[•][•][•] ← φ                                                      (8)
    L_a[•][•] ← φ                                                               (9)
    for each e in A begin                                                       (10)
        idxs ← i such that s(e) ∈ N_g[i]                                        (11)
        idxt ← i such that t(e) ∈ N_g[i]                                        (12)
        if merged_arc[idxs][idxt][fa(e)] = φ then begin                         (13)
            L_a[idxs][idxt] ← L_a[idxs][idxt] ∪ {fa(e)}                         (14)
            e.source ← mergedNode[idxs]; e.target ← mergedNode[idxt]            (15)
            mergedArc[idxs][idxt][fa(e)] ← e                                    (16)
        end                                                                     (17)
    end                                                                         (18)
    A ← ∪_{i=1}^{lnum} ∪_{j=1}^{lnum} ∪_{fa∈L_a[i][j]} mergedArc[i][j][fa]      (19)
    return (N, A)                                                               (20)
end
procedure getGluingNodeSet(N_1,...,N_k) begin
    N ← ∪_{i=1}^{k} N_i                                                         (21)
    lnum ← 0                                                                    (22)
    for each x in N begin                                                       (23)
        found ← FALSE                                                           (24)
        for i = 1 to lnum begin                                                 (25)
            if L_n[i] ∩ fn(x) ≠ φ then begin                                    (26)
                L_n[i] ← L_n[i] ∪ fn(x); N_g[i] ← N_g[i] ∪ {x}                  (27)
                found ← TRUE                                                    (28)
                break                                                           (29)
            end                                                                 (30)
        end                                                                     (31)
        if found =FALSE then begin                                              (32)
            lnum ← lnum + 1                                                     (33)
            L_n[lnum] ← fn(x); N_g[lnum] ← {x}                                  (34)
        end                                                                     (35)
    end                                                                         (36)
    return (lnum, L_n[•], N_g[•])                                               (37)
end
```

Figure 2.6: Procedure for Gluing

2.6 Examples

This section gives several examples of translation according to the PCT model.

Figure 2.7 shows how the Japanese sentence "Kanojo no me ga totemo kireina no wo shitteiru" is translated into the English sentence "(I) know that she has very beautiful eyes." In this figure, (a) is an input sentence structure, (b), (c), and (d) are translation patterns (precisely, pattern instances), and (e) is an output structure produced by PCT. In these patterns, mapping lines not marked $M\uparrow$ and $M\downarrow$ have both $M\uparrow$ and $M\downarrow$. Dotted lines designate matchings between nodes of the input structure and nodes of matching graphs of patterns, and numbers in braces denote node label-sets. In this example, we assume type hierarchies in which, for instance, *yoogen* (*predicate*) is a super-category of *keiyoo* (*adj*), and "kanojo" ("she") is an instance of *human*. Note that the node labels of both "have" in pattern instance (c) and the lower 'pred' in pattern instance (b) are changed from that of the corresponding Japanese word "kirei" ("beautiful") by the G_c node relabeling procedure.

Figure 2.8 shows how the Japanese coordinate conjunctive noun phrase "bideo to keisanki no kaisha" is translated into the English noun phrase "video and computer company." In this figure, (a) is an input sentence structure, (b), (c), and (d) are pattern instances matching the input, and (e) is an output structure produced by PCT. Note that (c) and (d) are instances of the same translation pattern, which matches a part of a coordinate conjunctive noun.

Figure 2.9 shows how an English sentence containing a relative clause, "The car which he bought is expensive," is translated into the Japanese sentence "kare ga katta kuruma wa takai." In this figure, (a) is an input dependency structure, (b), (c), and (d) are pattern instances, and (e) is an output structure.

2.7 Discussion

Several models have been proposed for the transfer process. They can be classified according to the following viewpoints: sequential or parallel production, destructive or non-destructive transducer, and data structure. *Sequential production* means that

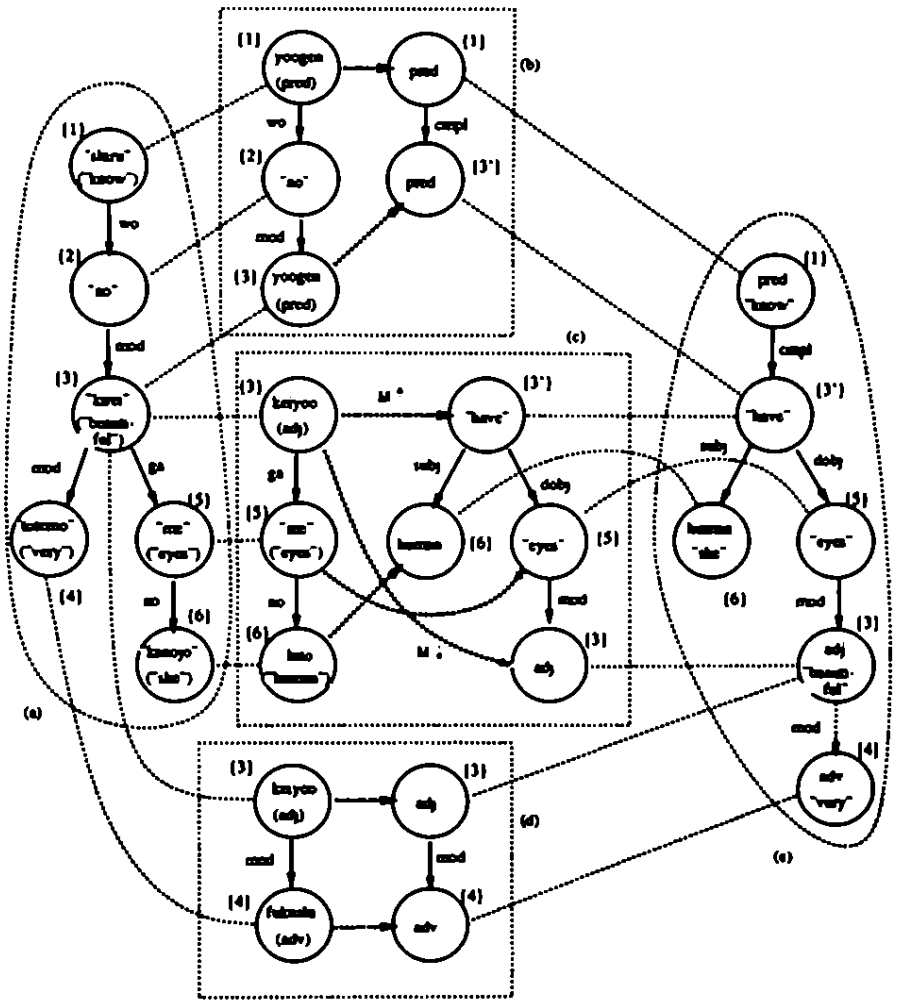

Figure 2.7: Example 1 of translation according to the PCT model

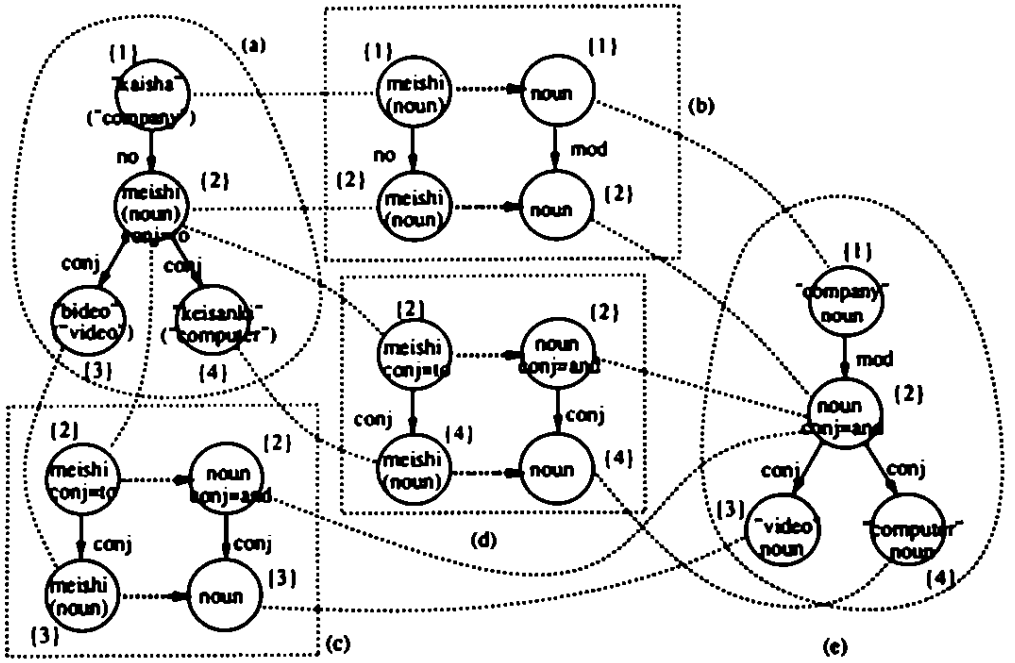

Figure 2.8: Example 2 of translation according to the PCT model

a part of the input structure is rewritten once and this process is continued until all parts have been rewritten. *Parallel production*, on the other hand, means that the whole input structure is rewritten by one application of rules. These terms are used in the area of Graph Grammars [Ehrig 79]. Further, a *destructive transducer* means that the input structure is rewritten destructively, whereas a *non-destructive transducer* means that the input structure is never destroyed during transduction. In this section, I will discuss several related studies with reference to the above points.

GRADE [Nagao and Tsujii 86] is a typical sequential destructive tree-to-tree transducer. It is a very powerful tree-rewriting system in which almost any kind of tree operation can be written in rules, and in which most rules are mutually related. ROBRA [Boitet and Nedobejkine 81] [Hutchins 86], developed in GETA, is also a tree-to-tree transducing system which has similar properties to GRADE such that rewriting rules are mutually related and rules are grouped as sub-grammars. In these systems, the application order of rules is critical. Therefore, rule writers must be responsible for both controlling the order of rule application and ensuring the production of a tree structure.

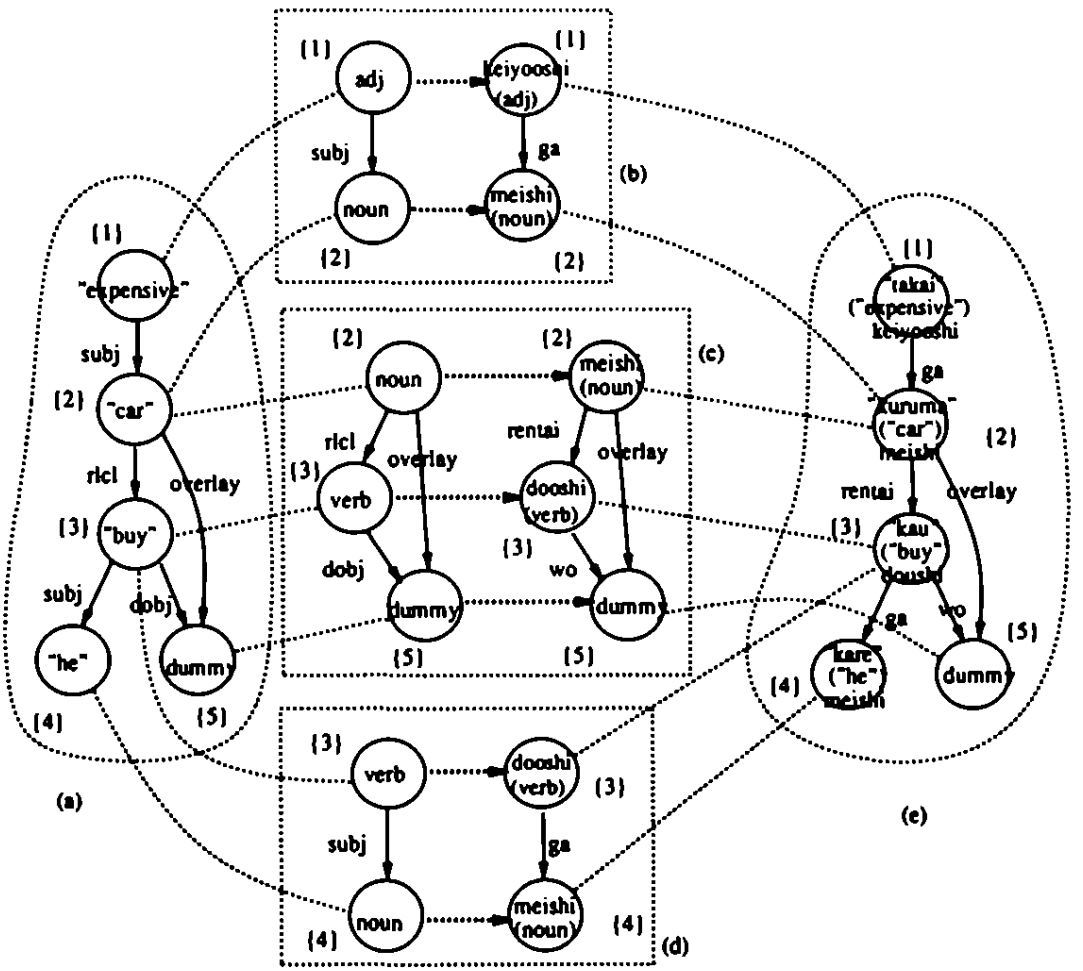

Figure 2.9: Example 3 of translation according to the PCT model

One of the problems of sequential destructive transfer systems is that they produce a data structure that holds features and grammatical relations of both source and target languages in the course of the transfer process. If a rule needs to refer to the features of a remote part of the input data, it is difficult to specify the part, because it may or may not have been translated into the target language. Rules must be ordered to avoid complicated specification. This makes the control of the application of rules very complicated. Thus, it can be said that non-destructive transfers have better properties than destructive ones in terms of ease of description and modularity of rules.

Lexical functional transfer (LFT) [Kudo and Nomura 86] is a sequential top-down non-destructive functional-RLDAG-to-functional-RLDAG transducer, which is closely related to the *PCT* proposed in this thesis. Rules are 3-tuples of source feature structure, target feature structure, and mappings between them, and an input feature structure is converted into an output feature structure starting from the root and proceeding down to the leaves. There is only one kind of mapping between a source node and a target node, and if a source node has an ancestor then the governing relation of their corresponding nodes in the target feature structure must be preserved. By using this restricted mapping, LFT ensures that a feature structure (functional RLDAG) is produced.

MBT-II [Sato and Nagao 90] was the first attempt to implement a fully example-based transfer system. Its transfer mechanism is a parallel non-destructive tree-to-tree transducer, and has the almost the same properties as LFT except that it uses a tree. MT by Lexicalized Tree Adjoining Grammar (or LTAG) [Abeillé et al. 90] uses a mechanism similar to the PCT, with paired derivation trees of source and target languages as translation rules. An input sentence is parsed by the source grammar, and at the same time, its output tree is generated by derivation pairs of trees used in the parsing. The ROSETTA system [Schenk 86] maps a syntactic derivation tree of a source sentence to a target derivation tree through a sort of interlingual semantic derivation tree. Further, the transfer module of EUROTRA [MTJ 91a] [MTJ 91b] generates a target interface structure (IS) by applying tree-to-tree translation rules to a source IS from the root node down to the leaves. These systems are sequential non-destructive tree-to-tree transducers adopting one-to-one correspondence.

In contrast, the proposed PCT is a parallel non-destructive RLDAG-to-RLDAG transducer employing $M\uparrow$ and $M\downarrow$ mappings between a source structure and a

target structure; these mappings are more flexible than those used in the systems mentioned above. In general, if a word in the source language is translated as several words in the target language, the translations of its governing and governed structures are not necessarily connected to only one of the words used to translate it in the target language. Therefore, we must designate the connections for those structures in the target language. As shown in Figure 2.4, one-to-one mapping cannot express the fact that "always" modifies "have," whereas $M\uparrow$ and $M\downarrow$ mappings can. For a system that employs one-to-one mapping to express this translation, it must have another mechanism for designating such connections, or a translation pattern that translates (a) directly as (c). Consequently, PCT has more expressive power and that its translation pattern database can be smaller than those of systems using one-to-one mapping. In example-based machine translation, whenever an input cannot be expressed as a combination of examples (translation patterns in PCT), a pair of the input and its translation should be added to the example-base as a new translation pattern. However, this leads to an explosion in the number of examples. To avoid this, we need an appropriate grain size for translation patterns.

PCT uses an RLDAG as a data structure. The main reason for this is that graphs are more flexible than trees for expressing translation patterns. In addition, recent grammar theories, such as LFG [Kaplan and Bresnan 82], GPSG [Gazdar et al. 85], and FUG [Kay 85], use feature structures (functional RLDAGs) as basic data structures, and RG [Perlmutter 83] [Perlmutter and Rosen 84] uses graph expressions (RLDAGs). PCT is intended to be used as a transfer mechanism in such systems. If an RLDAG is used, however, it is necessary to ensure that this parallel production produces an RLDAG. To preserve this RLDAG-to-RLDAG production, our model clarifies the conditions that given pattern instances should satisfy.[11]

Another salient feature of PCT is that it is independent of the pattern selection mechanism. Therefore, the most suitable pattern-selection mechanism for an application can be used together with PCT. In most conventional transfer systems, the pattern selection process and the transducing process are combined into a single process, and the pattern search/application order is fixed as, for example, top-down or bottom-up. In systems of this kind, the most appropriate pattern set is not necessarily found. On the other hand, systems using PCT can employ a pattern selection

[11] In real applications, the given RLDAGs do not necessarily satisfy the conditions. In this case, the next appropriate pattern set is tried, if the pattern set can be ordered according to some criteria by a pattern-selection mechanism.

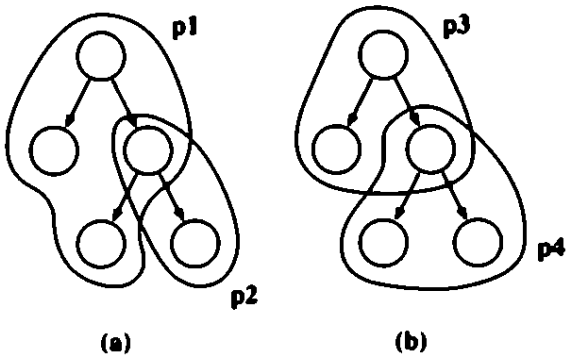

Figure 2.10: Different covers

mechanism that finds the most appropriate set of translation patterns on the basis of the total matching cost. In Figure 2.10, suppose that the matching cost of $p1$ is lower than that of $p3$, but that the matching cost of $p1 + p2$ is larger than that of $p3 + p4$. In this case, if the search order is fixed as top-down, then (a) is found to be the most appropriate pattern set, but if the total cost is considered, (b) is more appropriate.

2.8 Summary

In this chapter, I have proposed a new model, called Pattern Combination Transfer (PCT), of the transfer process used in machine translation systems. In this model, which is a parallel non-destructive RLDAG-to-RLDAG transducing system, we adopt extended mappings between the source and target parts of a translation pattern, and show that these mappings are more flexible and beneficial than one-to-one mapping, and that such translation patterns can be used bi-directionally.

To embed PCT into a practical machine translation system, it is necessary to attach a process that chooses an appropriate set of patterns for an input. Since PCT has no restrictions necessitating that the pattern selection mechanism should be based on similarity, any kind of selection method can be used with it. If PCT is coupled to a mechanism for selecting examples similar to an input, the system becomes an example-based transfer system. The next chapter describes an example-based Japanese-to-English transfer system, called *SimTran*, which incorporates this PCT

model with the Japanese similarity calculation method and the selection mechanism of appropriate translation patterns.

Chapter 3

Similarity-Driven Transfer System (SimTran)

3.1 Introduction

This chapter describes a Japanese-to-English transfer system called *similarity-driven transfer system (Simtran)* as an instance of an example-based transfer system using the PCT model.

As mentioned in the previous chapter, the PCT model does not deal with calculation of the similarity between the input and translation patterns, or with selection of translation patterns to be applied for the input. This chapter therefore also gives a method for calculating similarity in Japanese and a method for selecting an appropriate set of translation patterns.

The PCT model is intended to be used as a foundation of an example-based transfer system; however, this does not mean that existing transfer knowledge should be abandoned. Rather, such transfer knowledge should be used as a fail-safe mechanism if there are no appropriate examples. In the PCT model, both translation examples and existing transfer knowledge can be expressed as translation patterns, because PCT does not impose any restrictions on the contents of nodes. To deal with these translation patterns, of which some are derived from examples and some from transfer knowledge, the similarity calculation method of *SimTran* is designed to be

able to deal with nodes that have no lexical-forms.

There are many matchings of translation patterns on an input structure. Therefore, it is an important issue to select the most appropriate set of translation patterns to create a target structure. This chapter also proposes an efficient method to obtain those most similar to the input.

The next section defines the format of a translation pattern extended from the one described in Section 2.3. Section 3.3 presents a method for calculating the similarity between an input and the source part of a translation pattern. Section 3.4 describes a method for selecting appropriate translation patterns from a translation pattern database. Section 3.5 describes the flow of the *SimTran*. Section 3.6 gives examples of translation by *SimTran*, and Section 3.8 discusses several problems involved in the example-based approach, and outlines related work. Section 3.9 offers some concluding remarks on this chapter.

3.2 Translation Patterns

Since the tasks of a transfer system include lexicalization in a target structure, the translation pattern described in Section 2.3 is slightly modified to include the following mapping for lexicalization:

$$r = (G_m, G_c, M \uparrow\downarrow, M_L)$$

where G_m is a matching graph, G_c is a construction graph, $M \uparrow\downarrow$ is a structural mapping from G_m nodes to G_c nodes, and M_L is a lexical mapping from G_m nodes to G_c nodes.

The only modification is the addition of a **lexical mapping** M_L. The lexical mapping M_L designates lexical correspondences from a subset of G_m nodes to a subset of G_c nodes. In the construction of a target structure, there are cases in which a G_c node must be lexicalized by consulting a translation word dictionary, because G_c nodes may or may not have a lexical-form. In this case, this lexical mapping is used to retrieve the G_c node's source lexical-form.

Therefore, the conditions for valid translation patterns mentioned in Section 2.3 are modified as follows:

(1) $M\uparrow$, $M\downarrow$, and M_L are bijections,

(2) $M\uparrow(root(G_m)) = root(G_e)$, and

(3) let (a, b) be an element of M_L; then it is a member of either $M\uparrow$ or $M\downarrow$.

Condition (1) ensures that there is only one connection point in G_e for each mapping, and condition (2) ensures that the result generated by using this transfer model becomes a rooted graph (see the appendix for details). Further, condition (3) indicates that a lexical mapping must be a subset of a structural mapping.

3.3 Similarity Calculation

This section describes how the similarity between a translation pattern and an input is calculated.

There are many possible matchings between a given input dependency structure G_{in} and a translation pattern. Therefore, when the similarity between a G_{in} and a translation pattern is calculated, for each matching, a copy of a translation pattern (which we call a **pattern instance**) is created, and the similarity between the pattern instance and its projection on G_{in} is calculated.

3.3.1 Graph Distance

As a measure of the similarity between a G_m and its projection on G_{in}, we use the graph distance[1] between them. (The more distant two graphs are, the less similar they are.) The graph distance of a G_m and its projection is the sum of the distances between corresponding nodes. If there is a node in G_m that has no corresponding node in its projection, a penalty value is added.

[1] The distances discussed in this section are not actual distances in the mathematical sense.

3.3.2 Node Distance

When considering the distance between two words (nodes), we usually think of their semantic distance in a semantic hierarchy. This semantic distance is usually calculated according to lexical-forms that function as pointers to a semantic hierarchy. *SimTran*, however, can use a matching graph that does not have any lexical forms. Therefore, a node similarity must be calculated according to both lexical forms and syntactic features. In general, no matter what semantic hierarchy we use, it is inevitable that there will be some sort of distortion. Using both semantic distance and syntactic distance has the effect of remedying that distortion to a certain extent.

The node distance between a G_{in} node n_i and a G_m node n_m is defined as follows:

$$D_n(n_i, n_m) = \frac{D_f + D_s * \delta_s}{N_f + \delta_s}$$

where D_f is a feature node distance, D_s is a semantic node distance, N_f is the number of features in n_m for D_f, and δ_s is the weight of a semantic distance.

The semantic distance D_s between a G_{in} word w_{in} and a G_m word w_m is given by the following equation. In *SimTran*, *Bunrui Goi Hyou* [2] [NLRI 64] code (or *bghcode*) is used for calculating the semantic distance between two Japanese words.

$$D_s(w_{in}, w_m) = \begin{cases} 0 & w_{in} = w_m \\ 0.5 & w_{in} \text{ or } w_m \text{ is unknown} \\ 1 & w_{in} \text{ and } w_m \text{ are unknown} \\ \frac{|bgh(w_{in}) - bgh(w_m)| + \delta_b}{bghmax + \delta_b} & otherwise \end{cases}$$

where $bgh(w)$ is the fraction part of the bghcode of w, $bghmax$ is a constant that is the greatest difference between any two distinct bghcode fraction parts, and δ_b is a penalty incurred if two words are not identical.

The feature distance D_f between a G_{in} node n_{in} and a G_m node n_m is given as

[2] *Bunrui Goi Hyou* (or BGH) is a Japanese semantic hierarchy forming a tree structure, in which words are located in leaf nodes, and have a fraction number (or *bghcode*) indicating a semantic code. The integer part of the bghcode roughly corresponds to a syntactic category, and therefore only the fraction part of it is used.

follows:

$$D_f(n_{in}, n_m) = \sum_{f \in n_m} d_f(n_{in}, f)$$

$$d_f(n_{in}, fn : fv) = \begin{cases} 0 & f_{in}(fn_{in} : fv_{in}) \text{ whose } fn_{in} = fn, \text{ and} \\ & fv \text{ is consistent with } fv_{in} \\ 1 & \text{otherwise} \end{cases}$$

In the above equation, the consistency checking might depend on the application. For instance, unification [Knight 89], the ψ-term [Ait-Kaci 86], or some such mechanism might be suitable for the consistency checking. Further, the consistency checking mechanism might differ according to the features. Actually, in *SimTran*, the checking of the consistency of part-of-speech feature assumes a hierarchy of parts-of-speech, and two parts-of-speech are consistent if and only if one is an ancestor of the other in the hierarchy.

3.3.3 A Problem of Exceptional Translation Patterns

When calculating the similarity between an input G_{in} and patterns in a translation pattern database, we assume that each translation pattern has the same weight in the translation pattern database space. This can be considered by using the analogy of cells such as those shown in Figure 3.1. In this figure, a dot represents a translation pattern, and a cell represents the space in which an input is determined to be similar. There are, however, some translation patterns that should be used only for identical inputs, that is *peculiar* or *exceptional* translation patterns, such as colloquial or idiomatic expressions. Such translation patterns are indicated by shaded dots in Figure 3.1.

To handle this peculiarity, we introduce a distinction for lexical-form values: a single-quoted string is used for a peculiar one and a double-quoted string for a normal one. If both nodes to be calculated have double-quoted strings as lexical-form values, the equation for the semantic node distance given in the previous section is used. But, if either has a single-quoted string as a lexical-form value, the following equation is used as the semantic node distance:

$$D_s(w_1, w_2) = \begin{cases} 0 & w_1 \text{ is identical to } w_2 \\ 1 & \text{otherwise} \end{cases}$$

This equation means that the semantic node distance becomes 0 only when the lexical-forms of both nodes are identical; otherwise, it becomes 1.

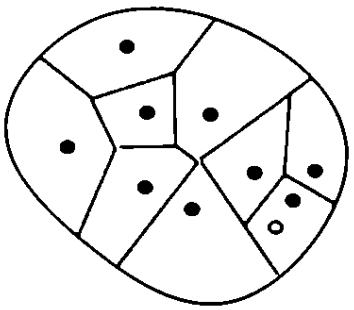

Figure 3.1: A translation pattern database space

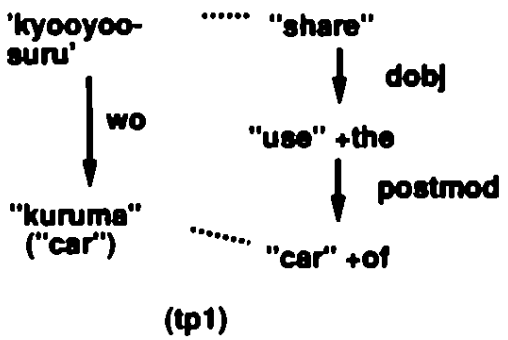

Figure 3.2: A translation pattern with a peculiarity

By using this distinction, it is possible to introduce a peculiarity into a part of a translation pattern. Colloquial or idiomatic expressions are expressed as translation patterns all of whose nodes have single-quoted strings as lexical-form values. But not all nodes in the source part of an exceptional translation pattern are necessarily single-quoted strings; single-quoted string nodes and double-quoted string nodes may be mixed in a translation pattern. For instance, Figure 3.2 shows an translation pattern that has a peculiarity in its source part; that is, the root node of the Japanese part is the only single-quoted string, and it matches only an input whose root node is 'kyooyoosuru.'

By using this distinction of lexical-forms, we can integrate exception handling into the similarity calculation framework without separating this task as a pre-process

or post-process.

3.4 Efficient Cover Search

Given an input rldag G_{in}, and a set of projections of translation patterns, as mentioned in Section 2.4, we must choose an i-cover on G_{in}. However, there can be several sets of projections that construct an i-cover on G_{in}. Therefore, we must select the most appropriate i-cover from among these candidates. This section describes an efficient method for selecting an i-cover.

3.4.1 Top-N Best Cover Search

Usually the best (minimum-cost) isomorphic cover to the input is sufficient, but, as we shall see in a later section, there are cases in which the second-best or third-best ones are required, because the similarity calculation described in the previous section does not reflect the grammatical or semantic accuracy when it is used in a target structure, and the accuracy becomes apparent only after a target structure has been constructed. Therefore, we need the best N i-covers in order to obtain the correct target structure. We have proposed an efficient cover search algorithm [Maruyama and Watanabe 92] that finds only the best i-cover. In this section, we will describe an extended version of the algorithm for obtaining the best N i-covers.

This algorithm assumes that an input structure is a tree, but in fact the input structure we use is an rldag. Therefore, when searching covers, an input structure must be expanded into a tree at nodes that have several ancestor nodes.[3] This expansion is, however, a trivial operation, because empirically there are few nodes with multiple ancestors in our application.

Before going to the details of the algorithm, we will define several notions. Let n be a node of the input, and C a cover; then n is said to be **closed** if and only if every outgoing arc of n is covered by the cover C[4], and n is said to be **open** if and only if

[3]We tried to extend this algorithm for a dag, but it became apparent that such an algorithm would require too much computation. Therefore we chose to expand an rldag as a tree before searching covers.

[4]In this context, an element x in a graph is said to be covered by a cover C if and only if x is

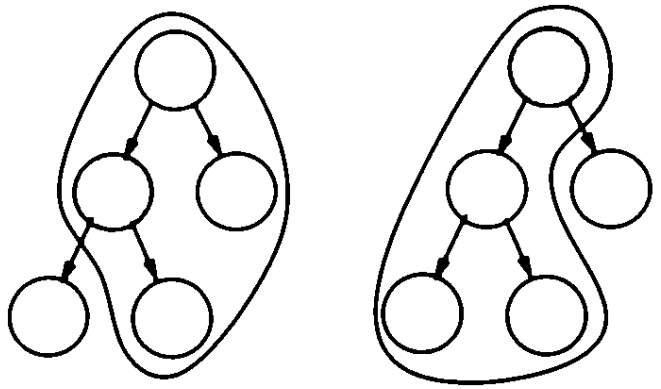

Figure 3.3: Well-formed and ill-formed covers

n is not closed. Further, a cover C is called **well-formed** if and only if it satisfies the following conditions:

(1) C has the only one root node.

(2) There are no open nodes in C other than the root node of C.

A cover that is not well-formed is called **ill-formed**. In Figure 3.3, shaded nodes represent open nodes; (a) is an ill-formed cover, because it violates the second condition, whereas, (b) is a well-formed cover.

The type of i-cover we are seeking is a well-formed cover containing all arcs going out from the root of the input. Therefore, given a projection (called a *base projection*) we must make a well-formed cover based on it. This process is briefly described as follows: *Given a base projection P whose root is n, if it has any descendant open nodes, for each open node x, a well-formed cover containing only arcs that are not covered by P is merged with P. This process is continued until there are no descendant open nodes of P.*

For this construction, each node has a table (called the minimum-cost well-formed

included in any projection of C.

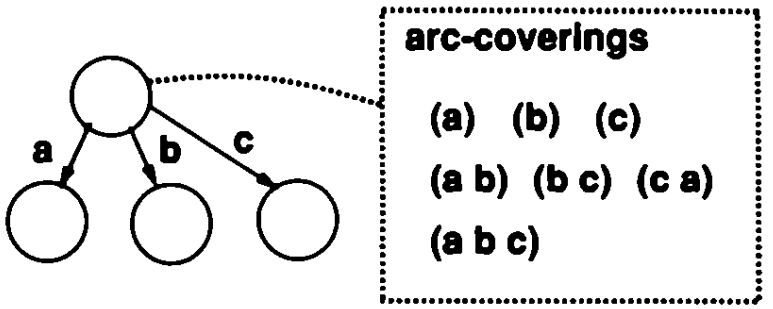

Figure 3.4: Arc-Coverings

cover table) which stores the minimum-cost well-formed cover for each element of the power set of outgoing arcs other than an empty set. Each element of this set is called an **arc-covering**. As an example, arc-coverings are shown in Figure 3.4.

The cost of a well-formed cover is the sum of the cost of a base projection and the costs of merged well-formed covers, where a distance is used as the cost of a projection.

The minimum-cost well-formed (or mc-well-formed) cover for an arc-covering is not necessarily constructed from a base projection covering arcs in the arc-covering. There are cases in which it is constructed from a combination of several mc-well-formed covers for disjoint subsets of the arc-covering. For instance, in Figure 3.4, the mc-well-formed cover for the arc-covering (a b) might be constructed from the mc-well-formed cover for the arc-covering (a) and the mc-well-formed cover for the arc-covering (b). Therefore, for an arc-covering whose branching factor is greater than 1, the costs of combinations of mc-well-formed covers for disjoint subsets of the arc-covering must be compared.

As a result, the best i-cover is obtained by making a well-formed cover of the root node of the input structure, which contains all outgoing arcs of the root. To obtain the best N i-covers, for each arc-covering the best N well-formed covers must be stored. In this algorithm, shown in Figure 3.5, M is the number of input nodes, N is the maximum number of well-formed covers stored in each arc-covering, and we assume that input nodes are numbered in post-fix order. For each node n, the number of arc-coverings is $2^{d(n)} - 1$ (where $d(n)$ is the branching factor of node n). Therefore, we provide an array $Cover[1..M][1..2^{d(n)}]$ for keeping information on the

well-formed covers and their costs.

The main loop consists of two sub-loops: one is the construction of well-formed covers of node n from the well-formed covers of descendant open nodes (lines 41-48), and the other is the merging of the well-formed covers of node n (lines 49-54). The procedure *product-cover* returns a list of pairs of a cover and its cost (called a cover list) constructed from a Cartesian product of the given two lists, and the procedure *get-topN* sorts the given cover list with respect to cost and returns the specified number of elements of the given list.

Let us consider the complexity of this algorithm. If we let p be the maximum number of projections, then the complexity of the first inner loop (lines 41-48) is $O(pMN^2logN)$, because the upper bound of the number of descendant open nodes x is M (line 44), and the complexity of the procedure *get-topN(product-cover())* is $O(N^2logN)$, because the maximum number of input elements for the procedure *get-topN* is N^2 (line 46). The complexity of the second inner loop (lines 49-54) depends on the branching factor of nodes in the input. We can assume that this branching factor of each node is a constant, because it is mostly limited to small integers in our application. Therefore, the number of combinations in line 52 is also a constant. Thus, the complexity of the second inner loop becomes $O(N^2logN)$. The overall complexity of this algorithm becomes $O(pM^2N^2logN)$. If only the best i-cover is sufficient, then the complexity is $O(pM^2)$, because the complexity of the procedure *get-topN(product-cover())* is $O(1)$ when N is 1.

3.5 Flow of SimTran

The previous two sections have described the two key methods in *SimTran*: the similarity calculation method and the efficient cover search method. This section describes the flow of the transfer processes of *SimTran*.

As shown in Figure 3.6, given an input G_{in}, for each sub-graph, possible projections are constructed by translation patterns, and the similarity values of these projections are also calculated by using the method described in Section 3.3. To reduce the computational cost, we restrict the translation patterns that can be applied for making a projection to (1) those whose root is adjacent[5] to the root node of the

[5]This is controlled by a parameter. In *SimTran*, two words are determined to be neighbors if

```
procedure topN-best-cover-search begin
    Cover[•][•] ← {}                                                          (38)
    for i := 1 to M begin                                                     (39)
      n := nodes[i]                                                           (40)
      for every projection p such that its root is n begin                    (41)
        D ← the arc-covering of the root of p                                 (42)
        Cover[n][D] ← ({p} cost(p))                                           (43)
        for every open node x in p other than n begin                         (44)
          D' ← the arc-covering of a well-formed cover such that x is closed  (45)
          Cover[n][D] ← get-topN(product-cover(Cover[n][D], Cover[x][D']), N)  (46)
        end                                                                   (47)
      end                                                                     (48)
      for k := 1 to 2^d(n) − 1 begin                                          (49)
        cover ← {}                                                            (50)
        for every i, j such that D_i ∩ D_j = φ and D_i ∪ D_j = D_k             (51)
          cover ← cover ∪ product-cover(Cover[n][D_i], Cover[n][D_j])         (52)
        Cover[n][D_k] ← get-topN(Cover[n][D_k] ∪ cover, N)                     (53)
      end                                                                     (54)
    end                                                                       (55)
end

procedure product-cover(cover1, cover2) begin
    cover ← {(cvr, cst) | ∃(cvr_1, cst_1) ∈ cover1, ∃(cvr_2, cst2) ∈ cover2,
          cvr = cvr_1 ∪ cvr_2, cst = cst_1 + cst_2}
    return cover
end
```

Figure 3.5: Algorithm of Top-N Best Cover Search

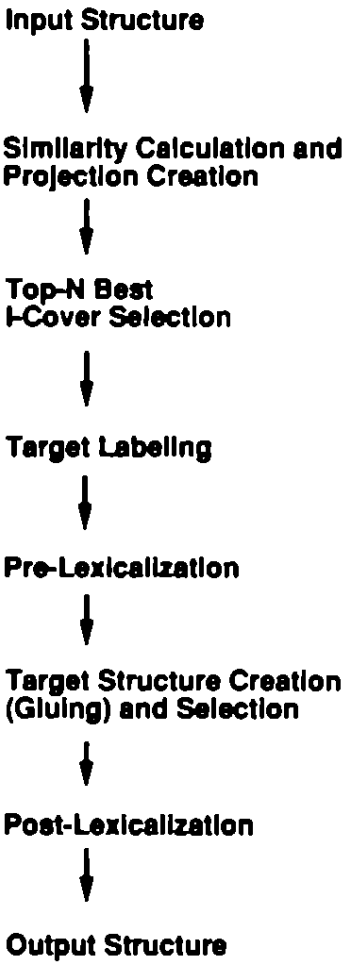

Figure 3.6: Flow of SimTran

current input sub-graph in the Bunrui Goi Hyou, and (2) those that do not have a lexical-form and whose part-of-speech is equal to or a ancestor of the part-of-speech of the root node of the current input sub-graph.

Next, the top-N best set of translation patterns is found by using the efficient cover search algorithm described in Section 3.4. When this algorithm is applied, the inverse of the similarity of a projection (or the graph distance) is used as a cost.

For each of the selected translation pattern sets, all nodes in the target side of translation pattern instance are labeled by using the labeling methods described in Section 2.5.1.

3.5.1 Pre-lexicalization

It may happen that a lexical-form of a G_e in the given pattern instance is not a candidate translation of the corresponding word in the input, because the lexical-form in the matching node in its G_m is not necessarily the same as the input word. In this case, such a node is lexicalized with candidate translation words, which are obtained by consulting a translation dictionary. A G_e node to be lexicalized is determined by the lexical mapping M_L.

3.5.2 Target Structure Creation and Selection

For each i-cover, after the G_es have been labeled and relabeled, the target structure is built by gluing the G_es. In *SimTran*, node merging in gluing always succeeds; consequently, if two features conflict, the feature whose pattern is more similar to the input is adopted.[6] The feature conflict, however, is reflected as a penalty for the similarity of the constructed structure, that is, the similarity of its i-cover pattern set. If there is a feature conflict, the next best i-cover set is tried, if it is more similar than the current i-cover pattern set. This is repeated until there are no more

they are in adjacent nodes in the Bunrui Goi Hyou hierarchy.

[6]This consistency checking depends on the features. For instance, part-of-speech feature values form a hierarchy, so that if the values to be checked are on a path in the hierarchy, they are consistent and the most specific value is used. Lexical-form feature values, however, are usually consistent only if they are identical.

similar i-cover candidates. By checking the similarity of the target structure, the most appropriate target structures can be selected.

3.5.3 Post-lexicalization

The constructed target structure may still be imperfect: there might be a G_e node that has no lexical-form, because some translation patterns might be syntax-level ones with no lexical-forms. Therefore, as in the pre-lexicalization phase, non-lexical G_e nodes are lexicalized with the translation words of their corresponding input nodes, which are determined by the lexical mapping M_L.

3.6 Examples

This section gives examples of translation by *SimTran*.

Figure 3.7 shows how the Japanese sentence "US ga ... wo fusegu tame ni buhin ni kanzei wo kakeru" is translated into the English sentence "US imposes tax on parts in order to protect ..." In this example, (a) is an input structure, (b), (c), and (d) are matched translation patterns (precisely pattern instances), and (e) is the output structure produced. The Japanese verb "kakeru" has several translation candidates associated with different governing words, as shown in Table 3.1. This table lists the top five similar patterns for the part "buhin ni kanzei wo kakeru" of the input. As shown in this table, pattern (c) is the most similar one. Note that this similarity calculation was done for all patterns, including syntax-level translation patterns. There were no appropriate example-based patterns for the part "US ga kakeru," and the syntax-level pattern (b) was therefore selected. Further, note that the lexical-forms in the *3 nodes of (c) and (e) are different, and that the *4 node of (c) has no lexical-form other than a preposition, whereas the *4 node of (e) has a lexical-form. The former was obtained by pre-lexicalization, and the latter by post-lexicalization.

Figure 3.8 shows another example in which the Japanese sentence "kare ga saisin no keisanki wo kau" is translated into the English sentence "he buys the latest computer." In this figure, (a) is an input dependency structure, (b), (c), (d), and

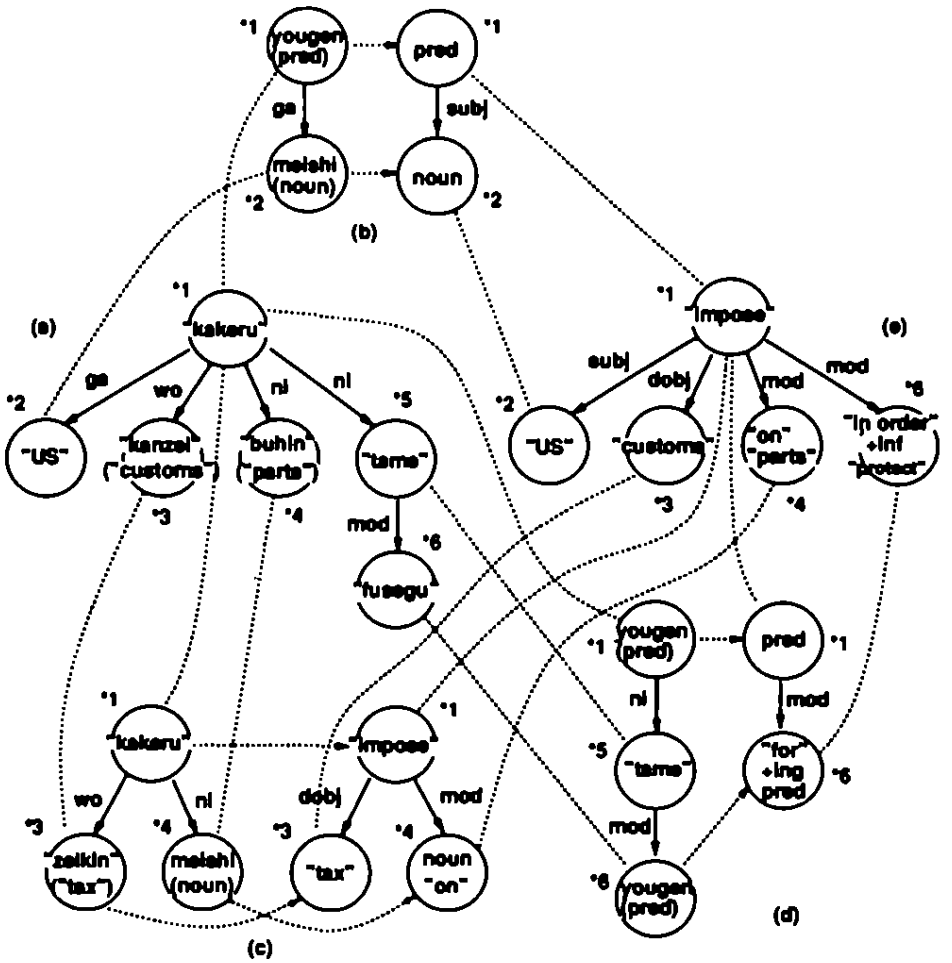

Figure 3.7: Example 1 of translation by *SimTran*

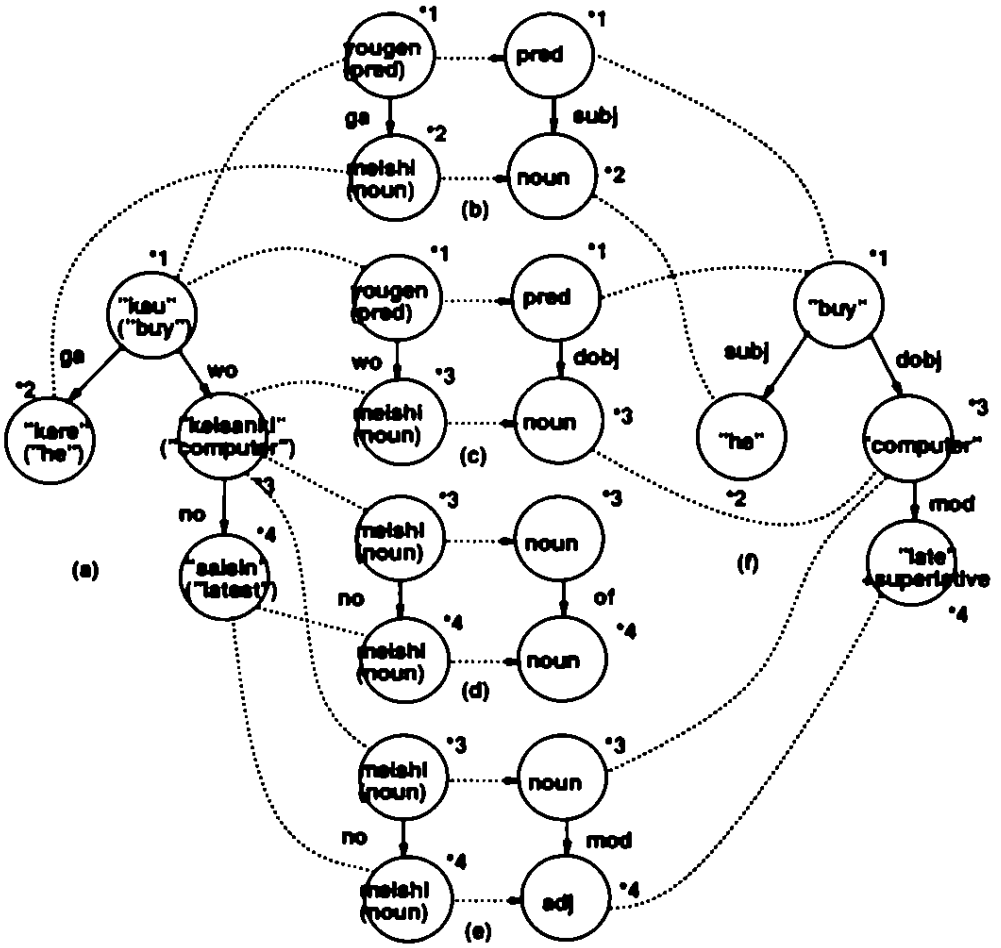

Figure 3.8: Example 2 of translation by *SimTran*

Similarity	Japanese-English
5.988	(meishi) ni zeikin wo kakeru impose tax on (noun)
3.077	(meishi) wo saiban ni kakeru take (noun) to court
2.717	(meishi) wo mado ni kakeru hang (noun) in window
2.545	(meishi) wo sutoobu ni kakeru put (noun) on stove
2.040	hankati ni kousui wo kakeru spray perfume on handkerchief

Table 3.1: Translation patterns and similarities for *kakeru*

(e) are matched translation patterns, and (f) is an output dependency structure constructed from (b) (c) and (e). Note that the source part "saisin no keisanki" has two matched translation patterns that have the same similarity with regard to the source structure, but that translation pattern (d) incurs a penalty in the construction of a target structure, because a noun translation of "saisin" does not exist. Therefore, if the translation pattern set {(b) (c) (d)} is applied first, it incurs a penalty in the construction of a target structure, and another translation pattern set {(b) (c) (e)} can successfully construct a target structure (f).

3.7 Implementation

SimTran is integrated as a transfer module into our Japanese-to-English machine translation system called *JETS*.[7] Figure 3.9 shows the system configuration of JETS. *SimTran* currently has about 20,000 translation patterns in its translation pattern database.

[7]JETS is a Japanese-to-English translation system that IBM Tokyo Research Laboratory has been developing for over 10 years.

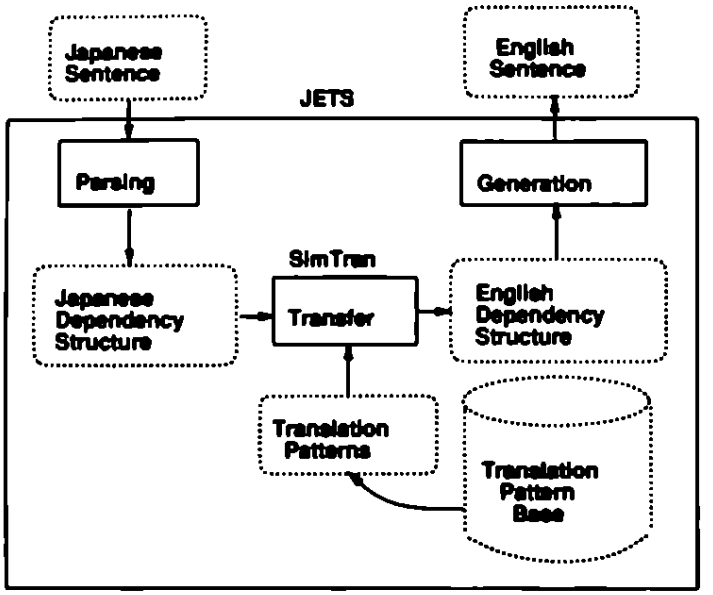

Figure 3.9: The system configuration of JETS

3.8 Discussion

As mentioned in Chapter 1, it is important to make patterns mutually independent, for reasons of maintenance, enhancement, and so on. In this respect, example-based processing is very promising. It is, however, still an emerging technology, and therefore some problems remain. In this section, we will point out several difficulties, and describe our approaches to managing them.

We have already discussed the peculiarity of the examples in Section 3.3.3, and will therefore not mention it again here.

In general, an example-based transfer system uses as transfer knowledge a pair of corresponding fragments of a source sentence and its translation, which have been more or less analyzed. Therefore, it is very difficult to collect a very large volume of transfer knowledge. If a pure example-based method, which uses just a pair of a source sentence and its translation sentence, is developed, collecting transfer knowledge will not matter, but such an approach has not yet been exploited. Therefore, we need (1) a method for collecting such transfer knowledge automatically from translation examples, and (2) a method for producing an output even if the volume of translation knowledge is not sufficiently large.

Since the former requirement will be dealt with in the next chapter, it will not be discussed here.

For the latter requirement, as Nagao pointed out [Nagao 92], a *hybrid system* that uses a conventional method and an example-based method complementarily is desirable. The following three forms can be considered as categories of such hybrid systems:

- **The Conventional-Method-First Hybrid System** applies a conventional method first and then applies an example-based method only for parts that cannot be dealt with by the conventional approach.
- **The Example-Based-Method-First Hybrid System** applies an example-based method first and then applies a conventional method for parts that cannot be dealt with by the example-based method.
- **The Seamless Hybrid System** unifies a conventional method and an example-based method seamlessly; that is, knowledge for both methods is used evenly.

The first category is very straightforward when an example-based approach is used in an existing conventional system. Several systems, such as that described by Uramoto [Uramoto 91], have been developed in this way. Transfer-Driven Machine Translation (TDMT) [Furuse and Iida 92] falls into the second category. On the other hand, *SimTran* conforms to the third category, because it can deal with example-based patterns together with syntax-level patterns uniformly, and manages to obtain an output even if there are not many example-based patterns, because syntax-level patterns are applied for parts that do not have enough similar example-based patterns.

The processing speed is a weak point of the example-based approach, because it requires an extensive search of the example base. Therefore, we must pursue efficient algorithms for enabling technologies, and efficient ways of reducing the search space. In this thesis, we have proposed an efficient algorithm for searching for an i-cover, and a merging method called *gluing*, which is an efficient and fast operation in comparison with unification. As for reducing the search space, generalization of patterns might be required in order to reduce the size of the transfer pattern database. Nomiyama has described one approach to generalizing of patterns [Nomiyama 92]. Further, we can expect that *massively parallel processing* will resolve this speed problem to some

extent [Kitano and Higuchi 91]. For instance, similarity calculation and PCT can be performed in parallel.[8]

Although there were several early experimental proposals [Nagao 84] [Sadler 89] and projects [Sumita et al. 90] [Sumita and Iida 91] [Furuse and Iida 92] related to EBMT, MBT-II [Sato and Nagao 90] is the first working prototype of an example-based transfer system, and demonstrates the promise of the EBMT approach. It uses Japanese-to-English translation examples as translation patterns, chooses the source trees of examples that are most similar to the input tree from the root node down to the leaves, and assembles those target trees to produce an output tree. As mentioned in the previous chapter, with respect to the transducing mechanism, MBT-II is a tree-to-tree transducer using one-to-one correspondence. Further, it differs from *SimTran* in that it uses only examples, and it dynamically constructs translation patterns from examples (or extracts relevant portions of examples).

As mentioned earlier, TDMT [Furuse and Iida 92] is a hybrid example-based translation system, which is intended to enable the example-based transfer process to take the initiative in translation: First of all, the example-based transfer system attempts a translation, then it passes any parts that it cannot deal with to the conventional MT system, and finally it combines both results. It differs from *SimTran* in the way in which it combines the conventional approach and the example-based approach.

In contrast, *SimTran* employs PCT, which is an rldag-to-rldag transducer using upward and downward correspondences. These extended correspondences are desirable for expressing the structural discrepancies that often occur in translation. Further, *SimTran* can use not only example-based translation patterns but also syntax-level translation patterns seamlessly.

3.9 Summary

In this chapter, we have described a transfer system, *SimTran*, that uses an example-based approach. Its translation pattern is simply a pair of parsed dependency structures in the source and target languages, and therefore both example-based

[8] PCT is called a parallel production system [Ehrig 79] that can produce an output structure in one execution of gluing if all the G_cs required to produce an output are supplied.

translation patterns and syntax-level translation patterns can be treated seamlessly. We showed that syntax-level translation patterns function as a fail-safe mechanism if there are no appropriate translation examples.

We also introduced two key components for an example-based transfer system: a similarity calculation method and a cover search method. The proposed similarity calculation method takes both semantic and syntactic features into consideration, and can deal with exceptional translation patterns and general translation patterns uniformly without destroying the whole framework of example-based processing. Further, the proposed top-N best cover search method can collect appropriate sets of translation patterns for an input efficiently. Some of these components are designed to be suitable for parallel computing.

Further, we pointed out several problems of the example-based approach, and ways of circumventing them. We believe that the example-based approach is a very promising technology for future machine translation systems, even though it poses several problems.[9] The ability to add new translation patterns with far fewer side effects than in conventional systems is very helpful for enhancing a translation system.

[9]Some of them are treated in later chapters.

Chapter 4

Automatic Extraction of Translation Patterns

4.1 Introduction: Automatic Creation of Translation Patterns

As mentioned in previous chapters, the example-based approach (EBA), an emerging technology proposed by Nagao [Nagao 84], has been extensively used in many areas of machine translation. A problem in applying the EBA approach to a transfer phase is that it is difficult and time-consuming to collect a large volume of translation patterns from translation examples, because such patterns are pairs of parsed structures in two languages, and are mostly constructed manually. Therefore, a method for collecting and constructing translation patterns automatically or semi-automatically is needed.

In considering the human task of acquiring translation patterns, we noticed that there are at least two different approaches: one is to find translation patterns by viewing many translation examples, and another is to find translation patterns by comparing a wrong translation and its correction. Several studies [Kaji et al. 92] [Utsuro et al. 92] [Matsumoto et al. 93] [Utsuro et al. 94] have followed the former approach. On the other hand, I have developed a system called **TranPet** that follows the latter; that is, one that can identify new translation patterns by comparing an MT result and its correction. In Figure 4.1, (a) is the dependency structure of

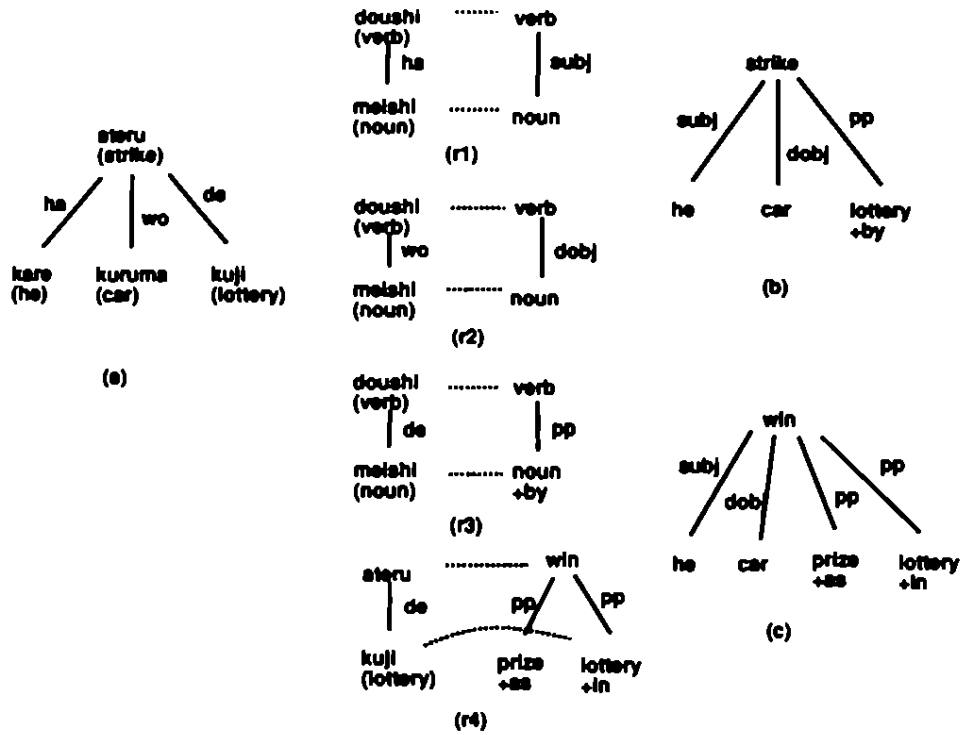

Figure 4.1: Example of extracting a new translation pattern

an input Japanese sentence, (r1), (r2), and (r3) are translation patterns used by a translation system, and (b) is the English dependency structure produced by these patterns. Clearly, (b) is not a correct translation of (a), whereas (c) is. **TranPet** compares the current output (b) and a correct output (c), and finds a translation pattern (r4).

Given an input string S_s, let S_t be an output string produced by a translation system TS, and let S_c be a correct translation of S_s. The purpose of the proposed method is to obtain translation patterns that produce the correct translation string S_c. We assume the following:

- TS is an example-based transfer system that can use translation patterns directly.
- Parsing is error-free; that is, the parsed structures of the source and target languages are correct.

- The translation pattern database includes some translation patterns.

The second condition might sound rather severe. However it does not mean that parsing must be done entirely by a parsing program, but rather that human supervision is allowed. Thus, we can break down the problem of obtaining translation patterns into the following two sub-problems:

- Making a mapping M_e from D_s to D_e,
- Finding translation patterns by comparing $\langle D_s, M_t, D_t \rangle$ with $\langle D_s, M_e, D_e \rangle$

where D_s is a dependency structure of an input string S_s, D_t is a dependency structure produced by TS, D_e is a dependency structure of S_e, and M_t is a mapping from D_s to D_t easily extracted from a set of translation patterns T_p used in constructing D_t by means of TS.

An overview of the **TranPet** system is given in the next section. Section 4.3 describes a method for finding mappings, and Section 4.4 a method for finding translation patterns. Section 4.5 gives an example of the system, and Section 4.6 discusses related work. Section 4.7 offers some concluding remarks.

4.2 System Flow

The system flow of **TranPet** is shown in Figure 4.2. An input Japanese sentence S_s is translated into an English sentence S_t by a Japanese-to-English machine translation system called *JETS*, which incorporates *SimTran* as a transfer module. If the translation is wrong, a user provides the system with a correct translation S_e, which is then processed by a target-language parser to give a dependency structure D_e. Subsequently, the correspondence (mapping data) between D_s and D_e is determined by the Mapper. Finally, the Pattern Extractor finds relevant translation patterns by comparing D_e and D_t with D_s, T_p translation patterns used when creating D_t, and mapping data.

TranPet is an interactive system, which requires human interaction in all the subtasks, because these tasks (parsing, finding mappings, and finding translation patterns) are not error-free.

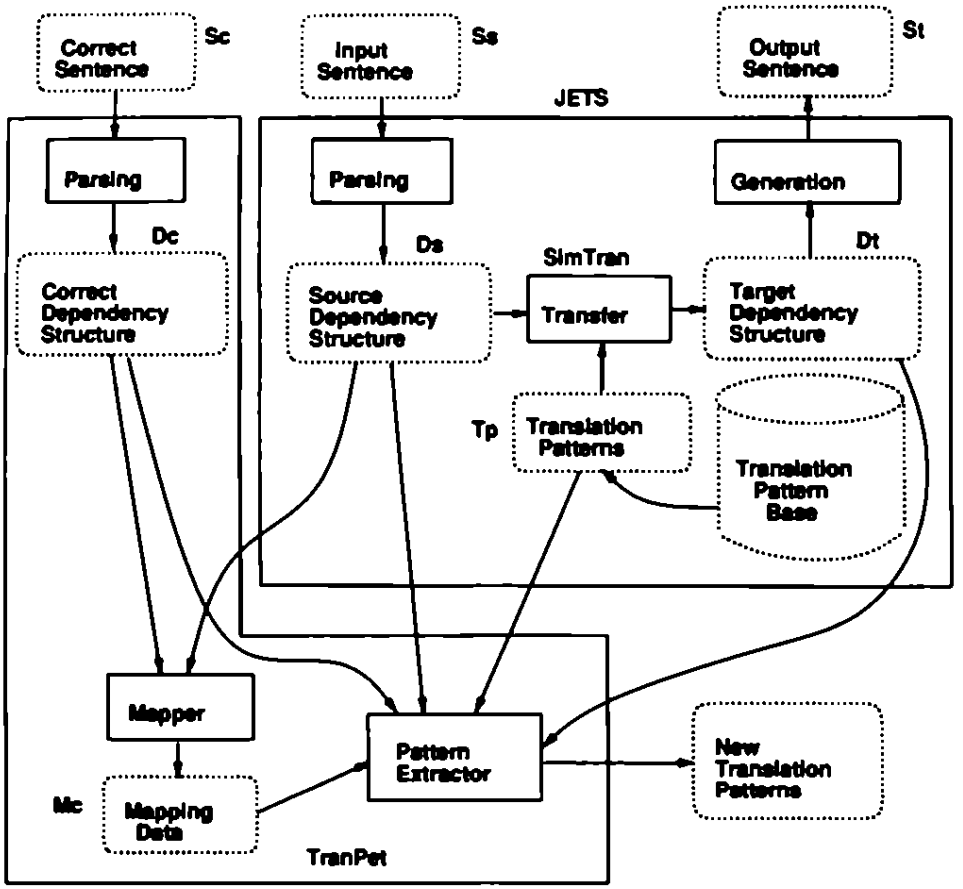

Figure 4.2: Flow of the system

4.3 Finding a Mapping between Source and Target

This section describes how to find mappings between the two dependency structures mentioned in the previous section.

4.3.1 Finding a Lexical Mapping

Utsuro et al. [Utsuro et al. 92] and Kaji et al. [Kaji et al. 92] proposed methods for finding the correspondences between translation pairs. The former method finds

```
procedure lexical-mapping(D_s, D_t) begin
    lexmap ← φ; openlist ← φ                                    (56)
    N_t ← nodes of D_t                                          (57)
    for each node s in D_s begin                                (58)
        ts ← getPossibleTranslationNodes(s, D_t)                (59)
        if |ts| = 1 then add (s, first(ts)) to lexmap           (60)
        else if |ts| > 1 then add (s, ts) to openlist           (61)
    end                                                         (62)
    for each (s, ts) in openlist begin                          (63)
        loop begin                                              (64)
            s' ← getNeighborNotVisited(D_s, s)                  (65)
            if s' = φ then break                                (66)
            t' ← t such that (s', t) ∈ lexmap                   (67)
            ts' ← getClosestNeighbor(D_t, ts, t')               (68)
            if |ts'| = 1 then begin                             (69)
                add (s, first(ts')) to lexmap                   (70)
                break                                           (71)
            end                                                 (72)
        end                                                     (73)
    end                                                         (74)
    return lexmap
end
```

Figure 4.3: Algorithm for finding a lexical mapping

a one-to-one correspondence between two feature structures of translation equivalents, and the latter method finds a one-to-one correspondence between two phrase structures. This one-to-one mapping corresponds to lexical mapping in *SimTran*.

Figure 4.3 shows an algorithm for finding lexical mapping. First, for each node, possible translation nodes are checked by using a source-to-target dictionary (line 59). If the translation node is uniquely determined, then a pair of the source node and its translation node is added to the M^L (line 60). For each node that has multiple possible translation nodes, a check is made to establish whether the translation node of one of its neighbor nodes has already been determined, and if so, the translation node nearest to that of the neighbor node is used (lines 63–74). The function $getNeighborNotVisited(D_s, s)$ returns a neighbor node of s in D_s which is not visited with regard to s, and the function $getClosestNeighbor(G_t, ts, t')$ returns

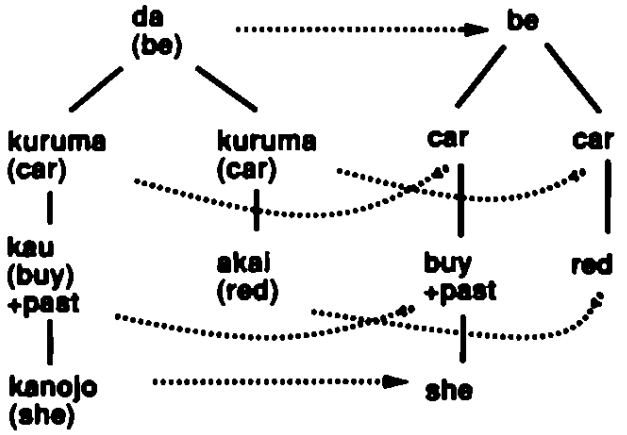

Figure 4.4: An example of finding lexical mapping

the node of ts that is the closest neighbor to t' in G_t.

Figure 4.4 shows dependency structures of a Japanese sentence "kanojo ga katta kuruma ha akai kuruma da," and its English translation "A car she bought is a red car." In this case, Japanese words "da," "kau," "kanojo," and "akai" are uniquely related to their corresponding words in English, but, neither instance of "kuruma"s is determined, since there are two instances of "car" in the English side. By using the above-mentioned algorithm, the left "kuruma" is related to the left "car," because the neighbor word "kau" is related to "buy" and the "buy" is nearer to the left "car" than the right "car." Likewise, the right "kuruma" is related to the right "car."

4.3.2 Finding a Structural Mapping

This section describes an algorithm for finding the upward and downward structural mappings introduced in PCT.

Figure 4.5 shows an algorithm for finding structural mapping. Briefly, this algorithm first finds a lexical mapping for a given translation pair (line 75), and then finds structural mappings (lines 78-99). Structural mappings are found as follows: A search is made for two correspondences (s_1, t_1) and (s_2, t_2) causing head switching such that s_1 is an ancestor of s_2 while t_1 is a descendant of t_2 (lines 78-81). If such

```
procedure mapping(D_s, D_t) begin
    M^L ← lexical-mapping(D_s, D_t) ar                                    (75)
    M ↑ ← M^L                                                             (76)
    M ↓ ← M^L                                                             (77)
    for any node pair s_1 and s_2 in D_s such that s_1 is an ancestor node of s_2 begin  (78)
      t_1 ← M^L(s_1)                                                      (79)
      t_2 ← M^L(s_2)                                                      (80)
      if t_2 is an ancestor node of t_1 then                              (81)
        remove (s_1, t_1) from M ↑                                        (82)
        t_3 ← t_1                                                         (83)
        for each n in ancestors of t_2 in D_t begin                       (84)
          if n is a root node, or n or one of its parent nodes
                    is related to by M ↑ or M ↓ then begin                (85)
            t_3 ← n                                                       (86)
            break                                                         (87)
          end                                                             (88)
        end                                                               (89)
        add (s_1, t_3) to M ↑                                             (90)
      end                                                                 (91)
      R_s ← root(D_s)                                                     (92)
      R_t ← root(D_t)                                                     (93)
      if M ↑(R_s) ≠ R_t then begin                                        (94)
        remove (R_s, M ↑(R_s)) from M ↑                                   (95)
        add (R_s, R_t) to M ↑                                             (96)
      end                                                                 (97)
    end                                                                   (98)
    return ⟨M^L, M ↑, M ↓⟩                                                (99)
end
```

Figure 4.5: Algorithm for finding a structural mapping

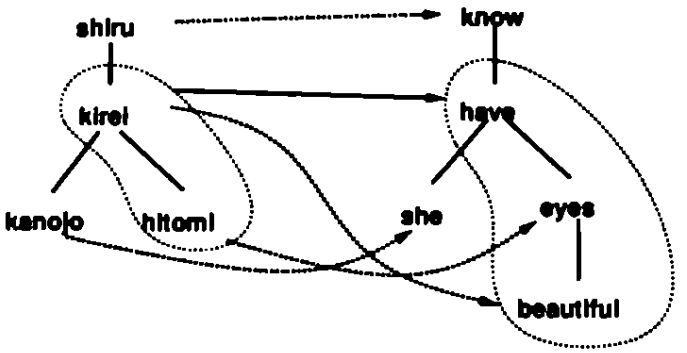

Figure 4.6: Example of finding structural mapping and phrasal correspondence

node pairs are found, the algorithm searches ancestors of t_2 for a node t_3 such that (1) t_3 is a root node, or (2) t_3 or one of its parent nodes is related to by another source node (line 85), and then makes (s_1, t_1) an downward mapping and (s_1, t_3) an upward mapping (lines 82-90). The last part (lines 94-97) checks whether both root nodes are related by an upward mapping, because a *SimTran* translation pattern must satisfy this condition. If they are not related, then it relates them.

For instance, in the case of Figure 4.6, dotted arrows gives the following lexical mappings:

M^L: {(shiru,know),(kirei,beautiful),(kanojo,she),(hitomi,eyes)}

In this example, head switching takes place for the node pairs (kirei,beautiful) and (hitomi,eyes), and these nodes hold true for lines 78-81. Therefore, (kirei,have) is found as an upward mapping. As a result, the following upward and downward mappings are obtained:

$M\uparrow$: {(shiru,know),(kirei,have),(kanojo,she),(hitomi,eyes)}
$M\downarrow$: {(shiru,know),(kirei,beautiful),(kanojo,she),(hitomi,eyes)}

4.3.3 Finding a Phrasal Mapping

For the sake of structural mapping, phrasal correspondence can be easily obtained by exploring the neighborhood of two distinct nodes projected by upward and downward mappings from one source node. For instance, in Figure 4.6, the source node "kirei" is related to the target node "have" by upward mapping, and is related to the target node "beautiful" by downward mapping. The minimum connected subgraph containing "have" and "beautiful" has another node, "eyes," which is also related to the source node "hitomi." This gives the phrasal correspondence between the minimum subgraph including "kirei" and "hitomi" and the minimum subgraph including "have," "eyes," and "beautiful" (both are surrounded by dotted lines).

The algorithm for finding phrasal correspondence is shown in Figure 4.7. In this algorithm, the function $relatedNodes(M\uparrow, M\downarrow, s)$ returns a set of nodes related from s by $M\uparrow$ or $M\downarrow$, and the function $nodesOfMinimumSubgraph()$ returns a set of nodes in the minimum subgraph consisting of the given nodes. Lines 104-109 find a set of source nodes and a set of target nodes such that mappings whose sources are those source nodes and mappings whose destinations are those target nodes are identical. These phrasal correspondences are limited in that they cannot include extra elements in the target part, that is, elements not related to any elements in the source part. This issue will be dealt with in the next section.

4.4 Extracting Valid Translation Patterns

This section describes how to find new translation patterns.

The basic idea for finding new translation patterns is to check whether each translation pattern used in MT is the same as the corresponding correct translation pattern. If they are not the same, then the correct one is a part of a new translation pattern. Therefore, for each translation pattern used in MT, we must construct a corresponding translation pattern in the correct translation. Since the source dependency structure is the same, we must identify the target part in D_c corresponding to a translation pattern used in MT.

Given a set of nodes in D_s, there is a set of nodes in D_t or D_c projected by M_t or M_c from those D_s nodes. We call the minimum connected subgraph containing all of

```
procedure phrasal-correspondence(D_s, D_t, M⌈, M⌊) begin
    phrases ← φ                                                          (100)
    for each pair of a node s_x and its child node s_y of D_s begin      (101)
        N_s ← {s_x, s_y}                                                 (102)
        if there is any element (N'_s, N'_t) in phrases such that N_s ⊆ N'_s
            then continue                                                (103)
        loop begin                                                       (104)
            N_t ← nodesOfMinimumSubgraph(∑_{s∈N_s} relatedNodes(M⌈, M⌊, s))   (105)
            if there is a D_s node s_z such that s_z ∉ N_s and
                    relatedNodes(M⌈, M⌊, s_z) ∩ N_t ≠ φ                   (106)
                then N_s ← N_s ∪ {s_z}                                   (107)
                else break                                               (108)
        end                                                              (109)
        add (N_s, N_t) to phrases                                        (110)
    end                                                                  (111)
    return phrases                                                       (112)
end
```

Figure 4.7: Algorithm for finding phrasal correspondence

these projected nodes in a target structure a **projected subgraph**. In Figure 4.8, the projected nodes of S1 and S2 are T3 and T4, respectively. Since the minimum connected subgraph including T3 and T4 must also include T2, a projected subgraph of D_s becomes a subgraph consisting of {T2, T3, T4}. Hence, for a translation pattern p used in MT, its corresponding translation pattern in the correct translation consists of the source part of p, a projected subgraph of p on D_c, and mappings between them (a subset of M_c).

Next, the equality of the two translation patterns must be checked. Given two translation patterns $P_i = (S_i, M_i, T_i)$ and $P_j = (S_j, M_j, T_j)$, P_i is identical to P_j if and only if the following conditions are all satisfied:

1. P_i and P_j are structurally identical.[1]

2. S_i and S_j are identical.[2]

[1] This means that node contents are ignored but arc labels must be checked.

[2] In this case, node contents are checked, but which features should be checked depends on the application. In our system, the lexical-form and part-of-speech are checked.

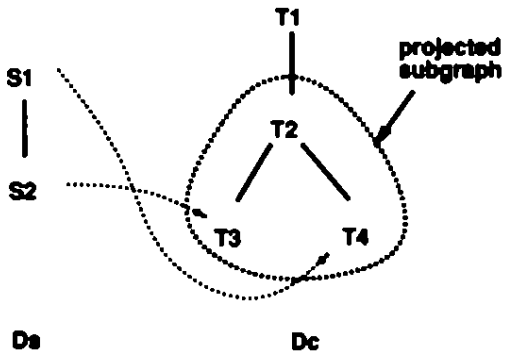

Figure 4.8: Example of projected subgraph

3. For each node s_i of S_i and its corresponding node s_j in S_j, let the node projected by $M \uparrow_i (s_i)$ be tu_i, let the node projected by $M \downarrow_i (s_i)$ be td_i, let the node projected by $M \uparrow_j (s_j)$ be tu_j, and let the node projected by $M \downarrow_j (s_j)$ be td_j; then tu_i subsumes tu_j and td_i subsumes td_j.[3]

An algorithm for finding translation patterns is shown in Figure 4.9. Lines 114-123 check, for each translation pattern p used to make D_t, whether the corresponding translation pattern p' is the same as p or not. If p' is not identical with p, then it is stored in the new translation pattern list; otherwise, it is stored in the same translation pattern list.[4] After that, the translation patterns in the new translation pattern list are merged if they share any common nodes (lines 124 129). Further, for each subgraph g of D_c that is included in neither the new translation pattern list nor the same translation pattern list, the minimum subgraph g' such that g' contains both g and any leaf node of p_t in the new translation pattern list is found, and g' is merged with p_t (lines 131-135).

The result of this algorithm may not be a complete translation pattern in the viewpoint of phrasal mapping. Therefore, a generated translation pattern by this algorithm is changed to be a phrasal mapping by using the algorithm described in Section 4.3.3 if it is an incomplete phrasal mapping.

[3]Since a node consists of several features, the subsumption relations on feature structures are used.

[4]Actually, mappings are not stored, because they are constructed later from a pair of source part and target part.

```
procedure find-trnpat(D_s,D_t,D_c,M_t,M_c,T_p) begin
    New ← φ, Same ← φ                                              (113)
    for each pattern p in T_p begin                                (114)
        p_s ← source part of p                                     (115)
        p_t ← target part of p                                     (116)
        m_t ← mappings between p_s and p_t                         (117)
        p_c ← projected subgraph of p_s in D_c                     (118)
        m_c ← mappings between p_s and p_c                         (119)
        if identical((p_s,m_t,p_t),(p_s,m_c,p_c))                  (120)
            then add (p_s,p_c) to Same                             (121)
            else add (p_s,p_c) to New                              (122)
    end                                                            (123)
    for each node n in D_s begin                                   (124)
        P ← {(p_s,p_c)|(p_s,p_c) ∈ New ∧ n ∈ p_s}                  (125)
        remove P from New                                          (126)
        (p'_s,p'_c) ← merge(P)                                     (127)
        add (p'_s,p'_c) to New                                     (128)
    end                                                            (129)
    D'_c ← merge(all p_ts in New and Same)                         (130)
    for each disconnected subgraph g in D_c − D'_c begin           (131)
        g' ← minimum subgraph of D_c that contains g and
                any leaf node x of p_t in New                      (132)
        remove (p_s,p_t) from New                                  (133)
        add (p_s,merge(p_t,g')) to New                             (134)
    end                                                            (135)
    return New                                                     (136)
end
```

Figure 4.9: Algorithm for finding translation patterns

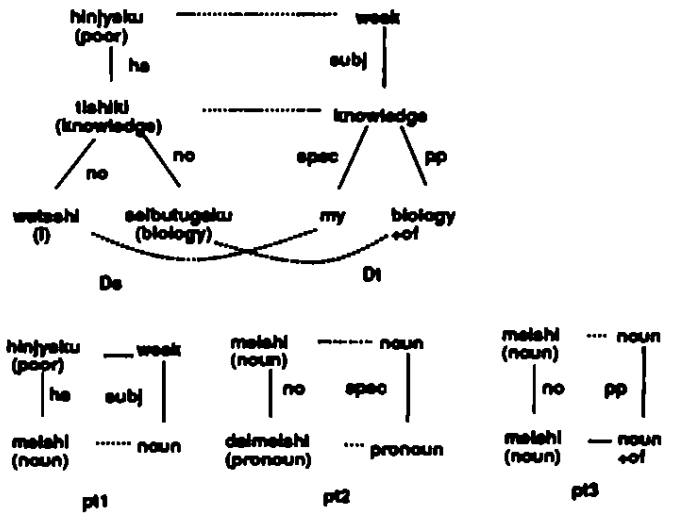

Figure 4.10: Translation patterns and dependency structures of input and translation by TS

4.5 Examples

This section gives an example of the use of this system to find new translation patterns.

In Figure 4.10, D_s is a dependency structure of the following Japanese sentence:

watashi no seibutugaku no tishiki ha hinjyakuda
(I have little knowledge of biology.)

and D_t is a dependency structure produced by TS as an output by using the translation patterns pt_1, pt_2, and pt_3. Dotted lines denote mappings between two structures. For convenience, an unmarked dotted line is equivalent to a line marked M^L, $M\uparrow$, and $M\downarrow$.

Suppose that D_c in Figure 4.11 is a correct English translation of D_s. Then the first step is to find mappings from D_s to D_c. By using the procedure described in the previous section, the mappings expressed by dotted lines in Figure 4.11 are obtained. In these mappings, note that the Japanese word "hinjyaku" ("poor") is related to the English word "have" even though it is not a translation word. This

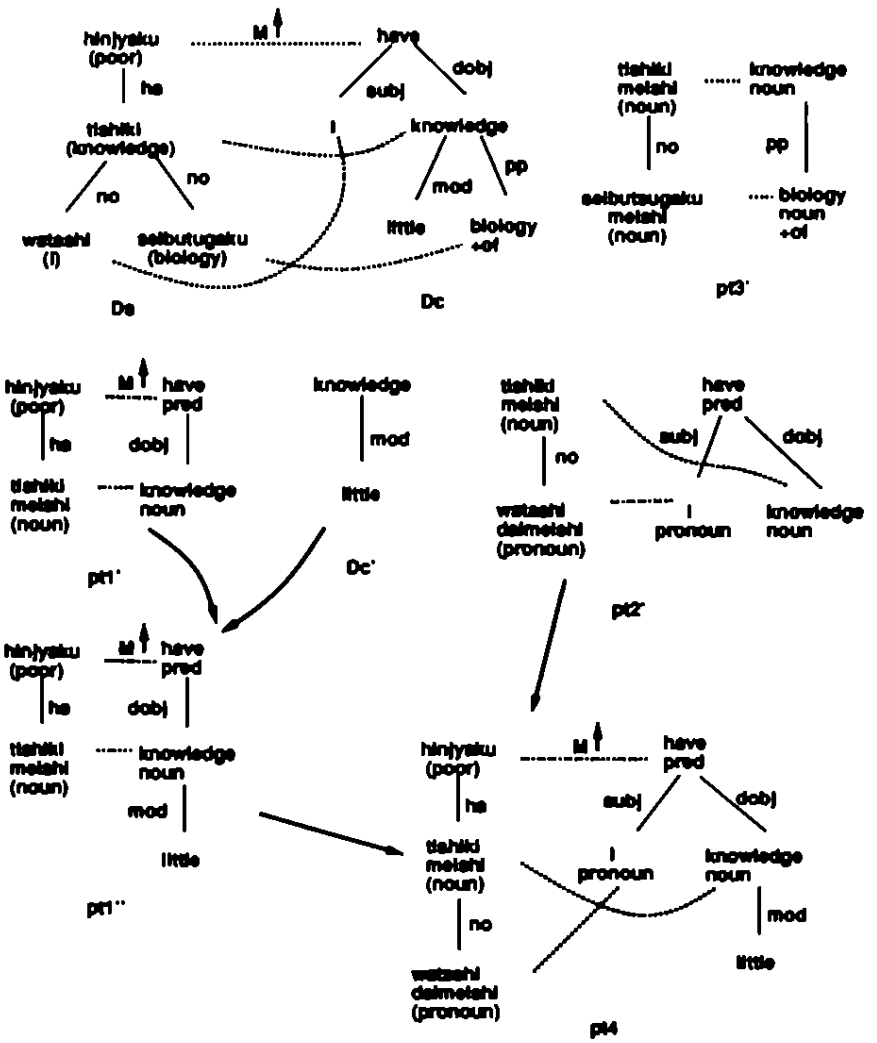

Figure 4.11: Translation patterns and dependency structures of input and correct translation

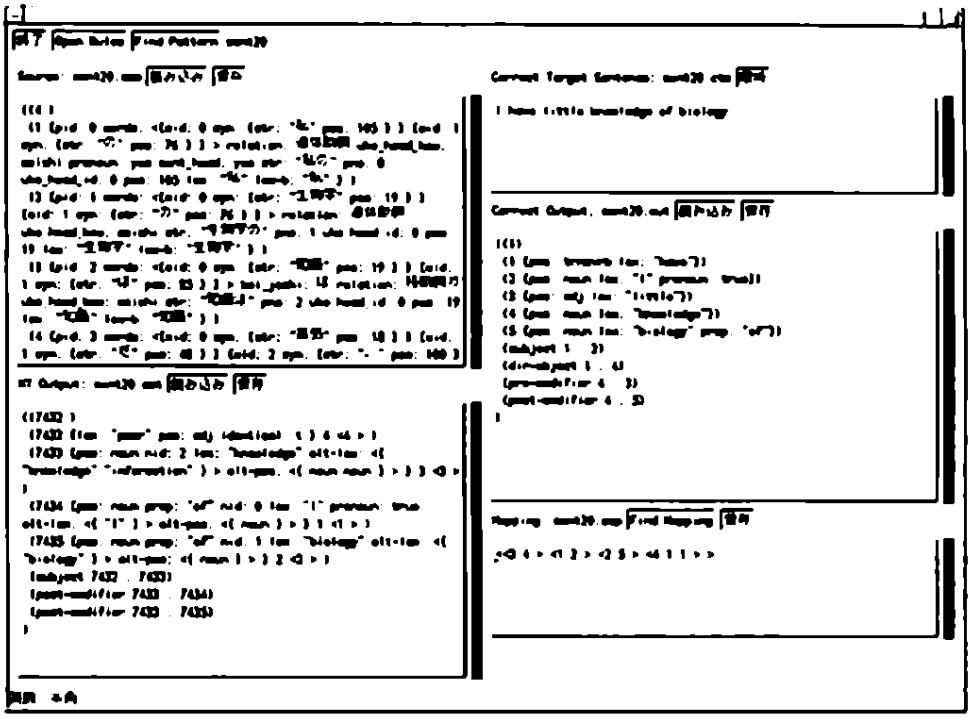

Figure 4.12: Screen image of TranPet

relationship can be established in the last part of the mapping algorithm.

The next step is to find structural differences between the translation patterns for $\langle D_s, D_t \rangle$ and for $\langle D_s, D_c \rangle$. The translation patterns corresponding to pt_1, pt_2, and pt_3 are shown as pt'_1, pt'_2, and pt'_3 in Figure 4.11. Comparison of these corresponding translation patterns shows that pt'_1 and pt'_2 are different, and they are stored in the new translation pattern list. D_c contains a portion, "little (mod)," that is not covered by the translation patterns pt'_1, pt'_2, and pt'_3. This portion is attached to pt'_1, and becomes pt''_1 in Figure 4.11. Finally, a new translation pattern pt_4 is obtained by merging pt''_1 and pt'_2 in the new translation pattern list.

The actual screen images of **TranPet** for this example are shown in Figures 4.12 and 4.13. In Figure 4.12, the source dependency structure appears in the upper left window, and the target dependency structure produced by *SimTran* is appears in the lower left window. The upper right window contains a correct English translation input by a user. When the *analysis* button is pushed, its parsed dependency structure appears in the middle window on the right. The *Find Mapping* button

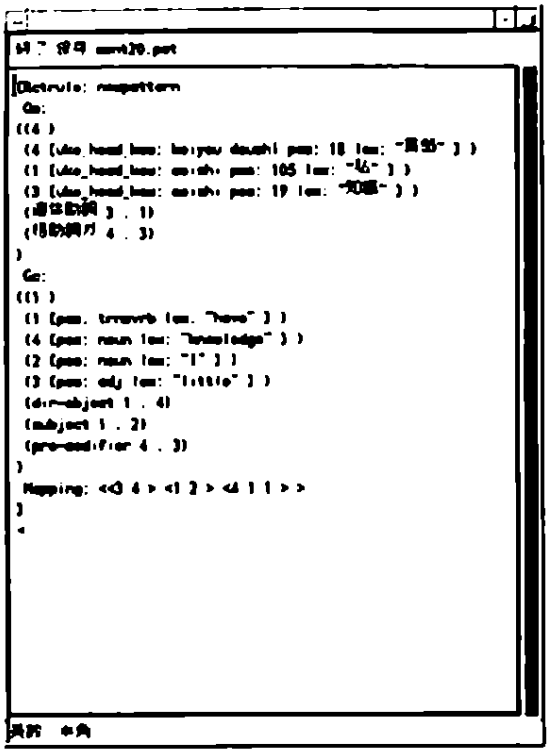

Figure 4.13: New translation pattern found by TranPet

executes the Mapper, and the mappings found by the Mapper appears in the lower right window. Finally, the *Find Pattern* button executes the Pattern Extractor, and the new translation pattern is displayed in another window, as shown in Figure 4.13.

4.6 Discussion

Strong recent interest in corpus-based processing has produced some results concerning the extraction of relevant information from bilingual corpora. There has been some research [Kaji et al. 92] [Utsuro et al. 92] [Matsumoto et al. 93] [Utsuro et al. 94] on finding correspondences between translation pair sentences, and extracting translation patterns from them. If we consider human processes for acquiring translation patterns, we can categorize them into two types: finding translation patterns by viewing many similar translations, and finding translation patterns by comparing a correct translation and a wrong translation. As an example of the former, we

can easily imagine that, given many translation instances of the Japanese word "toru" ("take"), one can obtain some translation patterns (or case frames) that are relevant for translating sentences containing "toru." The above-mentioned studies [Kaji et al. 92] [Utsuro et al. 92] [Matsumoto et al. 93] [Utsuro et al. 94] deal with the former type of process, whereas the method proposed in this chapter deals with the latter; that is, the method proposed in this chapter differs from previous ones in that it extracts new relevant translation patterns (which are not contained in the current translation pattern database) by comparing the result output by a translation system with a correct translation. The former type of method is necessary for creating a new translation pattern base from scratch; on the other hand, the latter type is effective for enhancing an existing translation pattern database in a bootstrapping manner.

Previous studies [Matsumoto et al. 93] [Utsuro et al. 94] have proposed a method for extracting not only one-to-one correspondence but also phrasal correspondence. As I mentioned in previous sections, our system also obtains structural correspondence; limited phrasal correspondence can be obtained in the mapping phase, and more global phrasal correspondence can be obtained by finding new translation patterns.

Some may feel that the proposed method is peculiar to *SimTran*. Admittedly, the method for finding structural mappings is unique to *SimTran*, but the method for finding new relevant translation patterns is applicable to any type of example-based transfer system, if lexical mapping is substituted for structural mappings in the descriptions above.

Since this system is an interactive tool that helps to create translation patterns, it is not easy to evaluate.[5] We have actually been using it for several months to create *SimTran* translation patterns. Our experience indicates that, if the mapping is correct, a result produced by the system can normally be used as a translation pattern after a little editing. Most errors in the mapping phase are caused by (1) a word used in a correct translation sentence not being registered in the dictionary as a translation for its source word, and (2) a single source word, such as a compound noun phrase, being translated as several target words. Further, creating a translation

[5]It is difficult to conduct an extensive test using many sentences, since the system depends heavily on the capability of the source and target parsers, and corrections are needed for the results of these parsers.

pattern by using this system takes half as much time as creating one from scratch. Therefore, this system greatly improves the productivity of pattern creation.

4.7 Summary

In this chapter, I have described a system called **TranPet** that compares a wrong translation and a correct translation in order to extract relevant translation patterns that should be added to a current translation pattern database. To find mappings between two parsed structures, I proposed a method for finding not only one-to-one correspondence but also structural mappings employed in *SimTran*. To find new translation patterns, I proposed a method for finding translation patterns from the differences between translation patterns used in a wrong translation and those used in a correct translation. This method is useful for extending or enhancing a current translation pattern database efficiently in a bootstrapping manner.

In future work, besides improving parsers, we must enhance the part of the system finding mappings, to reduce user interaction.

Chapter 5

Automatic Identification of Exceptional Translation Patterns

5.1 Introduction: Side Effects of Exceptional Translation Patterns

As mentioned in the previous section, one of the bottlenecks of example-based translation is the collection of large numbers of translation examples consisting of pairs of parsed structures in the source and target languages. By using some methods mentioned in the previous chapter, however, we can now collect translation patterns relatively easily.

There is another problem, though, called *example interference*, that an exceptional (or idiomatic) translation pattern is selected when a general translation pattern should be selected, and it has a side effect on the construction of a target structure. Suppose that we have the following two examples of translation from Japanese to English, (e1) and (e2),

(e1) watashi(I) ha konpyuutaa(computer) wo kyooyoosuru.
I share the use of a computer.

(e2) watashi(I) ha kuruma(car) wo tsukau.
I use a car.

and that we are given the following Japanese input sentence (s1):

(s1) watashi(I) ha dentaku(calculator) wo shiyoosuru.

In the above examples, (s1) is likely to be more similar to (e1) than (e2), because the three Japanese verbs "kyooyoosuru," "tsukau," and "shiyoosuru" are all very similar.[1] and "dentaku" ("calculator") is more similar to "konpyuutaa" ("computer") than "kuruma" ("car"). If this is the case, the English output obtained by using (e1) is (t1),[2] whereas it should be (t2):

(t1) I use the use of a calculator.

(t2) I use a calculator.

This problem occurs because example-based transfer systems choose examples simply on the basis of similarity. This can be considered by using the analogy of cells like those shown in Figure 5.1. In the figure, a dot represents a translation example, and a cell represents a space in which an input is determined to be similar. According to this analogy, an example-based system checks the cell in which an input is located, and uses an example governing the cell. If a new example is added in this space, a cell for it is created as in cell division. If an input happens to fall into the cell of an exceptional example, it is wrongly translated. There are two methods for avoiding this phenomenon: one is to treat an exceptional example as a special cell (a shaded dot of (a) in Figure 5.1) that has no extent in the example-base space, so that it cannot be used unless it matches the input exactly. The other is to surround an exceptional example with many general examples ((b) in Figure 5.1) so that it does not cause side effects. This is an ideal approach for example-based translation, but not a practical one, since it is very difficult to enumerate all general examples surrounding an exceptional example. Thus, the former method must inevitably be used; that is, an example-based translation system must deal with exceptional translation patterns separately when calculating similarity.

Therefore, we need a method for dealing with exceptional translation patterns in similarity calculation, and a method for identifying exceptional translation patterns

[1] Actually, they are in the same category (or the same leaf) in the Japanese thesaurus Bunrui-Goi-Hyou [NLRI 64].

[2] The main verb is changed from "share" to "use," because "share" is not a translation of "shiyousuru."

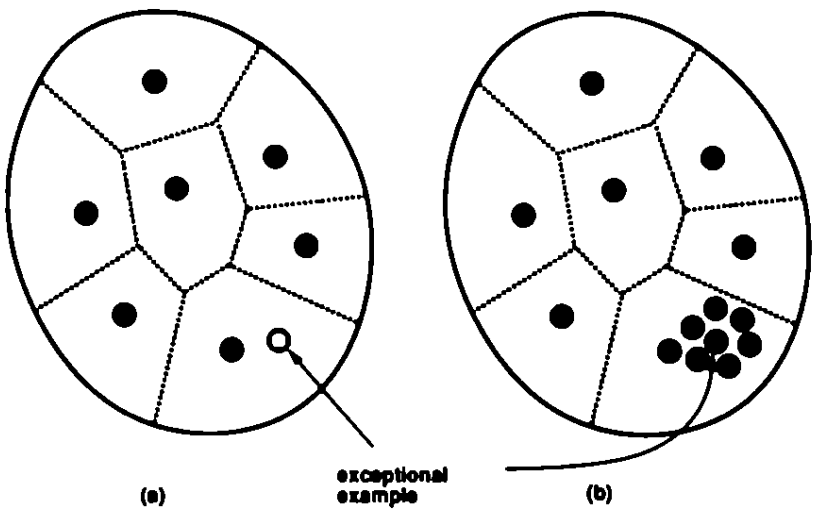

Figure 5.1: An example-base space

in a translation pattern database. Since a method for the former is described in Section 3.3.3, in this chapter we describes a method for the latter.

The next section describes a method for identifying exceptional translation patterns. Some experiments are reported in Section 5.3, and some issues are discussed in Section 5.4. Finally, Section 5.5 offers some concluding remarks.

5.2 Identifying Exceptional Translation Patterns

For most people, an exceptional translation pattern is likely to mean a pattern of translation for an idiomatic or colloquial expression. In general, an idiomatic translation pattern is a translation pattern whose target part is markedly different from those of translation patterns whose source parts are similar to that of the idiomatic pattern. From the viewpoint of the transfer process, what we would like to identify are translation patterns that may have side effects when they are selected instead of general translation patterns. We call such translation patterns **exceptional translation patterns**. According to this definition, exceptional translation patterns are not restricted to idiomatic patterns; in fact, the majority of translation patterns in this category are not idiomatic. Here, we classify exceptional translation patterns

into the following two categories:

- Extra-Exceptional Translation Patterns: These have some extra elements in the target part in addition to those in similar translation patterns.
- Intra-Exceptional Translation Patterns: These are almost the same as similar translation patterns, but several target words are different.

When exceptional translation patterns are found, it is important to know whether two translation patterns are equivalent or not. Therefore, **equivalent** translation patterns are defined as follows:

Given two dependency structures d_1 and d_2, then they are called **equivalent** if and only if they are structurally identical and corresponding nodes have similar semantic codes.[3] Further, two given translation patterns $tp_1 = \langle s_1, t_1, m_1 \rangle$ $tp_2 = \langle s_2, t_2, m_2 \rangle$, where s_i is a source part, t_i is a target part, and m_i is a mapping from s_i to t_i, are called *equivalent* if they satisfy the following conditions:

(1) Both source parts are equivalent, and both target parts are structurally identical.

(2) The roots of t_1 and t_2 are the same string.

(3) For each node n in s_1, $m_1(n)$ is one of the words used to translate n.

(4) For each node n in s_2, $m_2(n)$ is one of the words used to translate n.

The algorithm for identifying exceptional translation patterns is as follows:

Step 1 Divide translation patterns into several groups, each of which consists of equivalent translation patterns.

Step 2 For each pair of distinct translation pattern groups g_1 and g_2, if any pattern of g_1 is equivalent to any pattern of g_2 other than nodes governed by the root of the source part, then the translation patterns in g_1 and g_2 are marked as *general*.

[3]For instance, the semantic code in Japanese is Bunrui-Goi-Hyou code. The extent to which two words are determined to be similar is also a parameter. It may vary according to the system. In our system, two words are determined to be similar if they have the same semantic code.

Step 3 For each pair of distinct translation pattern groups g_1 and g_2, if the source part of any pattern (p_1) of g_1 is equivalent to the source part of any pattern of g_2, but their target parts are not structurally identical, because p_1 has extra elements, then the translation patterns of g_1 are marked as *extra-exceptional*.

Step 4 For each non-exceptional translation pattern group g_1, if there is another general translation pattern group g_2 such that any pattern (p_1) of g_1 is equivalent to any pattern of g_2 other than the root node in the target part of p_1, then the translation patterns of g_1 are marked as *intra-exceptional*.

Step 2 identifies possible general translation patterns if they are used in a relatively wide range of words, because in general an exceptional pattern is restricted in the usage of words. This approach, however, is not perfect for identifying general translation patterns, because there are cases in which the exceptionality derives from a single special word. Therefore, in the next step, checking does not exclude these possible general translation patterns. Step 3 identifies extra-exceptional translation patterns by checking the structure of the target part. Step 4 then identifies intra-exceptional ones by comparing the root node in the target part with the root nodes in the target part of possible general translation patterns. The reason why this comparison is restricted to possible general translation patterns is that intra-exceptional translation patterns have side effects only when they are similar to general translation patterns.

Figure 5.2 shows an example of the identification of exceptional translation patterns, in which the Japanese verbs "kyooyoosuru" and "tsukau" have the same bghcode, and the Japanese nouns "kuruma," "ongaku" have different bghcodes, whereas, "kuruma" and "jitensya," and "ongaku" and "mahoo" have the same bghcode, respectively. First, step 1 divides these translation patterns into four groups: group 1 consists of (tp1), group 2 consists of (tp2) and (tp3), group 3 consists of (tp4), and group 4 consists of (tp5). Step 2 identifies group 2 and 3 as general translation patterns, because "kuruma" and "ongaku" have different bghcodes. Subsequently, Step 3 identifies (tp1) as an extra-exceptional translation pattern, because (tp1) has the extra elements "the use of" for (tp2). Further, Step 4 identifies (tp5) as an intra-exceptional translation pattern, because (tp5) is equivalent to the general translation patterns (tp4) other than "use" and "practice" in the root nodes of the target parts.

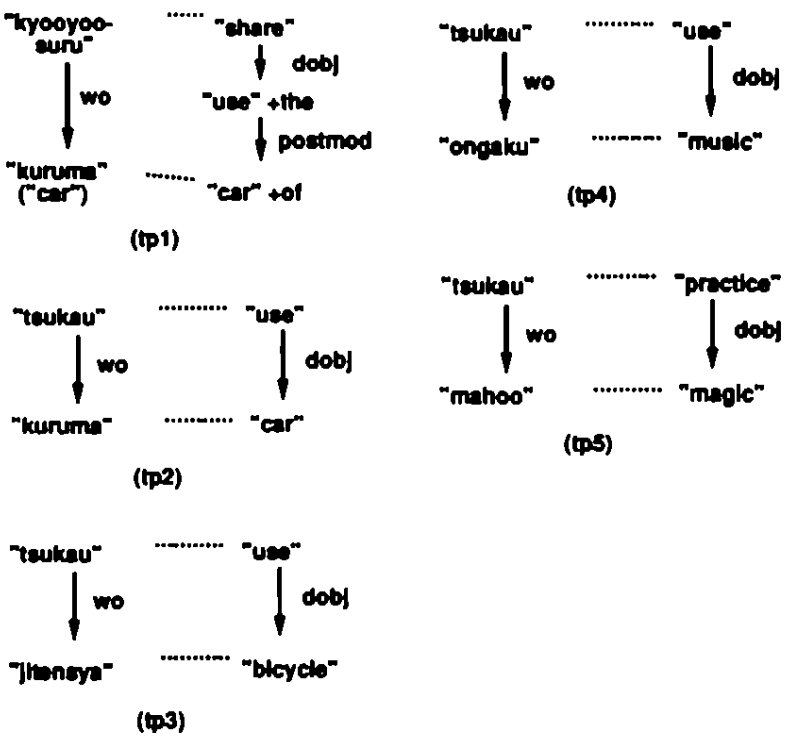

Figure 5.2: Example of the identification of exceptional translation patterns

Bghcode (example)	Total No. of Patterns	No. of General Patterns	No. of Exceptional Patterns (extra,intra)	Percentage of Exceptional Patterns (extra only)
15210(idousuru)	247	1	232 (228, 4)	93% (92%)
15270(iku)	174	0	138 (137, 1)	79% (78%)
15310(torikomu)	165	0	160 (150, 10)	96% (90%)
15600(tikazuku)	199	1	185 (178, 7)	92% (89%)
15710(kiru)	185	0	181 (159, 22)	97% (85%)
30110(kurushimu)	192	8	183 (160, 23)	95% (83%)
30200(suki)	280	6	271 (203, 68)	96% (72%)
30610(omou)	180	0	179 (169, 10)	99% (93%)
31200(iu)	191	0	173 (173, 0)	90% (90%)
36700(hattyuu)	182	0	181 (168, 13)	99% (92%)
38520(tsukau)	65	2	60 (53, 7)	92% (81%)

Table 5.1: Experimental results for transfer dictionary

5.3 Experiments

We have tested the above-mentioned algorithm with translation patterns in a Japanese-to-English transfer dictionary that was previously used in our rule-based MT system. For each bghcode, we collected translation patterns such that the root of the source part has the code, and applied the algorithm to the translation pattern set of each category. Table 5.1 shows the resulting top 10 categories with respect to the total number of occurrences. In most categories, more than 90% of translation patterns were identified as exceptional. The reason for the lopsidedness of this result is that the translation patterns described in the previous transfer dictionary were almost all exceptional cases that could not be dealt with by the default procedures coded in the transfer module. Therefore, this result indicates that the algorithm is able to identify exceptional translation patterns correctly.

5.4 Discussion

In conventional transfer systems [Nagao and Tsujii 86], transfer rules are roughly divided into general ones and exceptional (or idiomatic) ones. Such transfer systems usually check the exceptional cases first, and if these rules do not match the input, then apply general rules. On the other hand, example-based transfer systems deal with translation patterns (or examples) uniformly on the basis of similarity, according to the example-based principle. This mechanism causes the example interference problem. A very useful property of the example-based approach is that it allows a sentence and its translation to be added as an example if it cannot be translated correctly. However, if the same input as the newly added example is given, but when the resolution of the similarity calculation is not high enough, an input that is similar to but not exactly the same as the added example may not be translated correctly, because there may be another similar example that is exceptional. Therefore, it is very important to identify whether an example is general or exceptional.

After application of the algorithm described in this chapter, translation patterns are classified into the following categories: general, exceptional (extra- and intra-), and neutral. Neutral translation patterns, which are not marked general or exceptional, are translation patterns that do not have side effects. They are not used for a wide variety of words in the current translation pattern database. If more translation patterns are added later, they may be identified as general or exceptional. By this method, one can enable the system to identify exceptional translation patterns automatically by adding some general translation patterns similar to them. This is a very useful feature for bootstrapping of a translation pattern database. A weak point of this algorithm, however, is that it requires a large number of translation patterns. If enough translation patterns are not given, exceptional translation patterns may not be identified. However, collecting many translation patterns is no longer a serious problem, since several methods for collecting them automatically have been proposed in recent studies [Kaji et al. 92] [Utsuro et al. 92] [Matsumoto et al. 93] and in the previous chapter.

The method proposed in this chapter probably does not comply with human intuition regarding idiomatic translation patterns; rather, it detects translation patterns that are idiomatic for the system, in other words, patterns that might have side ef-

fects in the current set of translation patterns. It probably requires deeper semantic processing to identify translation patterns that are idiomatic in the conventional sense. It is more important for the system to identfy exceptional translation patterns which may have side effects than to identify ones which conforms to human sense of idiomatic expression, except for special cases.

5.5 Summary

In this chapter, I have discussed a problem of example-based transfer systems, *example interference*, that the selection of inappropriate translation patterns eventually leads to a wrong translation. To avoid this problem, we need a method for identifying exceptional translation patterns that may have side effects on the target side. I have proposed a method for distinguishing exceptional translation patterns from general translation patterns. In some cases, this method gives results that do not match human intuition regarding idiomatic translation patterns, but it can detect, from the viewpoint of example-based processing, translation patterns in the current translation pattern database that might have side effects.

This phenomenon, example interference, is also known as rule interference in conventional rule-based translation systems. Even if we adopt the example-based approach, this problem still exists when there are not enough translation patterns. How to collect sufficient quantity of translation patterns is an open problem. However, note that in the example-based approach, this phenomenon can easily controlled by using the proposed method; in the conventional approach, on the other hand, it is hard to control, because many rules are mutually related, and it is difficult to find how and when a rule should be blocked.

Chapter 6

Summary and Conclusions

The transfer phase in machine translation system has received less attention than the parsing and generation phases, and has a much less detailed theoretical background. Therefore, most conventional transfer systems consist of a huge network of mutually dependent transfer rules. This makes it difficult to enhance and maintain the transfer system, because it is very difficult to predict and block side effects that may be caused by adding new rules. One of solution to these problems is to hide human intuition and to use a sort of objective criterion in the rule selection phase. This is one of principles of the example-based approach (EBA). By adopting the EBA in the transfer phase, we can separate rule writing and rule selection; that is, we use similarity calculation in place of the rule-writer's intuition. If the similarity calculation can be properly designed, the EBA system has the great merits that if a sentence is not translated properly, we can get a correct translation by simply adding a translation pattern derived from the sentence and its correct translation, and that the side effects of new translation patterns are fewer than in conventional transfer systems.

Chapter 2 proposed a basic framework for the transducing system, called *pattern combination transfer* (PCT). This framework is well suited to the example-based approach. The PCT model, given some translation patterns that are pairs of dependency structures in the source and target languages and correspondences among related nodes, constructs a target structure by combining the target sides of the given translation patterns. The correspondence used in a translation pattern of the PCT model is not just a one-to-one mapping; rather, it is extended to have an up-

ward mapping and a downward mapping so that it can express peculiar translations of the type mentioned in Section 2.3.

Chapter 3 described an example-based transfer system called *SimTran*, which incorporates the PCT model with a similarity calculation method for Japanese, and a method for selecting appropriate translation patterns for the input. The similarity calculation method proposed in this chapter calculates the similarity of two given dependency structures in which the nodes are a set of features, and does not impose the restriction that a node must have a lexical-form. Therefore, the method can deal uniformly with translation patterns derived from translation examples and from conventional transfer knowledge, so long as they can be expressed as pairs of dependency structures. It can also deal with exact matching for exceptional translation patterns. Further, a method for collecting translation patterns can find the best set of translation patterns for the input from the viewpoint of the total cost of similarity. But, this similarity only reflects the source sides, and the final validity can be determined only after a target structure has been constructed. For cases in which the best translation pattern set does not create a valid translation, the selection method can maintain the top-N candidates of the translation pattern set.

Adoption of the EBA simplified the transfer system and overcame some problems in the transfer phase. However, other problems still remain, including the high cost of producing translation patterns, for which parsing is needed in order to create dependency structures. It is essential to reduce this cost. Chapter 4 dealt with the automatic creation of translation patterns, and proposes a method for creating translation patterns by comparing an example-based MT result and its correct translation. This method compares translation patterns used in a correct translation with translation patterns used in an example-based MT system, and creates new translation patterns by finding the changes needed to produce a correct translation. It is not a completely automatic process, since it requires human intervention in some phases such as paring, if there are ambiguities. This, however, is not a drawback, as it makes it much easier for humans to make translation patterns. In future, it may be possible for an end user to add new translation patterns to the translation pattern database by using this kind of tool.

Some other problems arise when there are not enough appropriate translation patterns for an input. How many translation patterns should be provided? The answer to this depends on the target domain of the translation and the expectations of

users, and the question is probably open. Therefore, we must create an EBMT system that can produce an appropriate output even in such a situation. One solution to this problem is to use syntax-level translation patterns used in conventional transfer systems as a fail-safe mechanism. As mentioned in Chapter 3, the PCT model does not require translation patterns to be constructed from a translation example, and the similarity calculation method in *SimTran* can deal with translation patterns derived from conventional transfer knowledge that do not have any lexical-form.

However, there still remains the problem of *example interference*, that exceptional translation patterns cause side effects in translation. Chapter 5 dealt with this problem and proposed a method for identifying exceptional translation patterns in the current translation pattern database. The method can identify general translation patterns that will not have side effects, and exceptional translation patterns that may have side effects. It detects a translation pattern that is exceptional in that there is a possibility that it will have side effects. Therefore, exceptional translation patterns detected by this method may be different from idiomatic translation patterns in the human sense. But it is more important to detect translation patterns that might cause problems for the system than idiomatic translation patterns in the human sense. Further, as mentioned in Chapter 3, the exception is expressed as a single-quoted string in a translation pattern, and the similarity calculation method of *SimTran* can seamlessly deal with these exceptional translation patterns by performing exact matching for single-quoted strings. Note that example interference is relatively easy to control by the proposed method in the example-based approach, whereas it is difficult to control in the conventional transfer approach.

In summary, this thesis has proposed a foundation model of the transfer phase called PCT, that is suited to the example-based approach, and has described an example-based transfer system called *SimTran*. Further, it has proposed a method for creating translation patterns in order to reduce the cost of creating translation patterns, and a method for identifying exceptional translation patterns in the current translation pattern database in order to improve the translation quality even in situations where the database does not contain enough translation patterns.

Since a machine translation system is a sort of simulation of the processes in a human brain and the behavior of the brain still not been fully explored, even in the field of medical science, it is almost impossible to express all the possibilities or situations in the translation as rules. A system which tries to deal with such a task

must be flexible so that new knowledge can be easily added when necessary. The example-based approach is more suitable than conventional approaches in this sense, and has the advantage that knowledge necessary for translation can be accessible to end users; that is, end users can directly supply translation examples as translation knowledge. The future direction of example-based translation will be to enhance this capability, that is, to pursue a complete example-based approach that minimizes user interactions in creating translation patterns from translation examples.

Appendix A

Conditions for Producing an RLDAG

In this appendix, we will describe the conditions in which PCT can produce an RLDAG.

Before going into the details, let us introduce several terms:

Definition A.0.1 (dumbbell) *A* **dumbbell** *is a node-arc-node triad* $(nx\ a\ ny)$ *such that* $nx = s(a) \wedge ny = t(a)$ *or* $nx = t(a) \wedge ny = s(a)$, *where* nx *and* ny *are nodes and* a *is an arc.*

Definition A.0.2 (node path, arc path) *A* **node path** *is an ordered set of dumbbells such that* $ny_i = nx_{i+1}$. *If we are not concerned with arcs, then a node path* $((n_1\ a_1\ n_2)\ \ldots\ (n_{n-1}\ a_n\ n_n))$ *is written as* $(n_1\ \ldots\ n_n)$. *An* **arc path** *is a node path such that each dumbbell* $(nx\ a\ ny)$ *satisfies* $nx = s(a) \wedge ny = t(a)$. *Since all nodes in a dumbbell are determined uniquely by its arc, an arc path* $((n_1\ a_1\ n_2)\ \ldots\ (n_{n-1}\ a_n\ n_n))$ *is simply written as* $(a_1\ \ldots\ a_n)$.

Definition A.0.3 (matching pivot) *Given some projections, we call a node contained in intersections of any two distinct projections a* **matching pivot**. *An origin node of a matching pivot is also called a matching pivot.*

Definition A.0.4 (gluing pivot) *Given some G_c's g_i, for any node n_i of g_j, n_i is called a* **gluing pivot** *if there is a node n_k of g_l ($i \neq k$ and $j \neq l$) such that $fn_j(n_i) \cap fn_l(n_k) \neq \phi$.*

Definition A.0.5 (gluing node) *If two distinct nodes n_i and n_j are gluing pivots such that $fn(n_i) \cap fn(n_k) \neq \phi$, then n_i is called a* **gluing node** *of n_j, and vice versa.*

Definition A.0.6 (gluing node set) *Let R_{gn} be a relation on a set of gluing pivots such that $nxR_{gn}ny$ if and only if ny is a gluing node of nx or if nx is identical with ny, and let R^*_{gn} be the reflexive and transitive closure of R_{gn}. Then for a gluing pivot n, a set $\{x | nR^*_{gn}x\}$ is called a* **gluing node set**, *and a union of labels of nodes in a gluing node set is called an* **identical label set**.

Definition A.0.7 (virtual node path, virtual arc path) *A* **virtual node path** *over some LDGs is a list of dumbbells $((nx_1\ a_1\ ny_1)...(nx_n\ a_n\ ny_n))$ such that $fn(ny_i) \cap fn(nx_{i+1}) \neq \phi$, where nx and ny are nodes and a is an arc of one of the LDGs.*
A **virtual arc path** *over some LDGs is a virtual node path, each dumbbell of which satisfies $nx = s(a) \wedge ny = t(a)$.*

In general, a graph produced by gluing RLDAGs is an LDG, so we need to find the conditions in which several RLDAGs may be glued to form an RLDAG. To prove that the resulting graph is rooted and acyclic, we must consider the conditions that RLDAGs have to satisfy.

First, let us consider the conditions for acyclicity.

Theorem A.0.1 (acyclicity) *A glued graph of several RLDAGs becomes an acyclic LDG if they satisfy the following conditions:*

(1) The nodes of each given RLDAG are labeled uniquely.

(2) Let P be a union of arc paths for each given RLDAG such that both ends are gluing pivots. Then there are no cycles in virtual arc paths constructed from elements of P.

[Proof of Theorem A.0.1]

Assume that there is a cyclic path in a glued graph.

Since there is no cyclic virtual arc path containing gluing pivots (condition (2)), it is obvious that there is no cyclic arc path including any glued node (which was a gluing pivot) in a glued graph. (p1)

Therefore, there must be a cyclic path that contains only nodes that are not gluing pivots. Since a node that is a gluing pivot in an RLDAG cannot make a path without gluing pivots to a node in another RLDAG, there must be a cyclic path within an RLDAG. From condition (1), there is no possibility of making a cyclic path in an RLDAG by gluing. (p2)

From (p1) and (p2), the assumption is wrong. Hence, if some RLDAGs satisfy these conditions, a glued graph of them becomes an ldag. □

A procedure for testing acyclicity is shown in Figure A.1. The function *getGluingNodeSet'* is the same as *getGluingNodeSet*, which is used in the gluing procedure, but it returns gluing node sets containing only gluing pivots. Lines 138 to 143 collect arc paths whose ends are both gluing pivots for each given RLDAG, where we suppose that the given RLDAGs have a table containing an ancestor-descendant relationship for every node pair. The subsequent part searches nodes for cyclic paths in depth-first order. Since each node has a state representing whether it has been visited, is being visited, or has not been visited, the search's encountering a visiting node implies that there is a cyclic path. The procedure *visit* is the main routine in searching for a cyclic path. It searches child nodes for a node that has not been visited and, for each node, this search is executed recursively until it reaches a leaf node or a visiting node.

Let us consider the complexity of this procedure. Let k be the number of given RLDAGs, let n be $|max(N_1, ..., N_k)|$, and let a be $|max(A_1, ..., A_k)|$. The complexity of the function *getGluingNodeSet'* is $O(k^2n^2)$. Since the maximal value of *lnum* (line 139) is kn and line 140 iterates at most k^2n^2 times, the complexity of the part collecting arc paths (lines 139-143) is $O(k^3n^3)$. On the other hand, in the part for searching for a cyclic path, the function *visit* is executed exactly once for each arc path in A, because it is a depth-first search. The maximal value of A is k^2n^2, because there may be an arc path between all pairs of gluing pivots in the worst case. Therefore, the complexity of the part for searching for a cyclic path (lines

```
procedure testAcyclicity(G1 = {N1, A1}....Gk = {Nk, Ak}) begin
    (lnum, Ln[], Ng[]) — getGluingNodeSet'(N1, ...., Nk)                    (137)
    A — ∅                                                                   (138)
    for i = 1 to lnum begin                                                 (139)
      for each distinct gluing node x and y in Ng[i] begin                  (140)
        if x is an ancestor of y then A — A ∪ {(x, y)}                      (141)
      end                                                                   (142)
    end                                                                     (143)
    N — ∪_{i=1}^{lnum} Ng[i]                                                (144)
    for each x in N, flag[x] — NOTVISITED                                   (145)
    for each x in N begin                                                   (146)
      if flag[x] =NOTVISITED then begin                                     (147)
        rc — visit(x, A, lnum, Ng[])                                        (148)
        if rc =FALSE then return FALSE                                      (149)
      end                                                                   (150)
    end                                                                     (151)
    return TRUE                                                             (152)
end
procedure visit(n, A, lnum, Ng[]) begin
    i — j such that n ∈ Ng[j]                                               (153)
    for each x in Ng[i], flag[x] — VISITING                                 (154)
    for each (x, y) in A such that x ∈ Ng[i] begin                          (155)
      if flag[y] =VISITING then return FALSE                                (156)
      if flag[y] =NOTVISITED then begin                                     (157)
        rc — visit(y, A, lnum, Ng[])                                        (158)
        if rc =FALSE then return FALSE                                      (159)
      end                                                                   (160)
    end                                                                     (161)
    flag[n] — VISITED                                                       (162)
    return TRUE                                                             (163)
end
```

Figure A.1: Procedure for testing acyclicity

146 151) is $O(k^2n^2)$. Hence the total complexity is $O(k^3n^3)$.

Next, we must consider conditions that produces a rooted LDG.

Theorem A.0.2 (rootedness) *A glued graph of several RLDAGs g_i becomes a rooted LDG if the RLDAGs satisfy at least one of the following conditions:*

(1) All root nodes are gluing pivots, and there is exactly one gluing node set whose elements are all roots.

(2) There is exactly one root node that is not a gluing pivot, and all other root nodes (which are gluing pivots) have at least one gluing node that is not a root node.

[Proof of Theorem A.0.2]

Nodes except for roots in given RLDAGs never become roots in a glued graph, because they have at least one governing node and this relationship is never changed in a glued graph. Therefore, we consider the following cases of roots:

(a) All roots are gluing pivots.

(a-1) If all the roots have any gluing node that is not a root, then a glued graph has no root, because all nodes have a governing node.

(a-2) If there is only one gluing node set whose elements are all roots, then a glued graph has only one root, because roots that are contained in other gluing node sets have a governing node and cannot be roots in the glued graph.

(a-3) If there is more than one gluing node set whose elements are all roots, then a glued graph has two or more roots, because they have no governing node in a glued graph.

(b) Only one root is not a gluing pivot.

(b-1) If all other roots have at least one gluing node that is not a root, a glued graph has only one root, because the other roots never become roots in the glued graph.

(b-2) If all gluing nodes of any other roots are roots, then a glued graph has more than one root, because these other roots become roots in the glued graph.

```
procedure testRootedness1(G_1, ..., G_k) begin
    (lnum, L_n[], N_g[]) ← getGluingNodeSet                          (164)
    cnt1 ← cnt ← 0                                                   (165)
    for i = 1 to lnum begin                                          (166)
        num ← |N_g[i]|                                               (167)
        froot ← fnotroot ← TRUE                                      (168)
        if num = 1 then begin                                        (169)
            if root(N_g[1]) =TRUE then return FALSE                  (170)
        else begin                                                   (171)
            fallroot ← TRUE                                          (172)
            for each x in N_g[i] begin                               (173)
                if root(x) =FALSE then begin                         (174)
                    fallroot ← FALSE                                 (175)
                    break                                            (176)
                end                                                  (177)
            end                                                      (178)
        end                                                          (179)
        if fallroot =TRUE then cnt1 ← cnt1 + 1                       (180)
        if cnt1 > 1 then return FALSE                                (181)
    end                                                              (182)
    return TRUE                                                      (183)
end
procedure testRootedness2(G_1, .., G_k) begin
    (lnum, L_n[], N_g[]) ← getGluingNodeSet                          (184)
    for i = 1 to lnum begin                                          (185)
        num ← |N_g[i]|                                               (186)
        if num = 1 then begin                                        (187)
            if root(N_g[i]) =TRUE then cnt2 ← cnt2 + 1               (188)
        end                                                          (189)
        else begin                                                   (190)
            froot ← fnotroot ← TRUE                                  (191)
            for each x in N_g[i] begin                               (192)
                if root(x) =TRUE then froot ← TRUE                   (193)
                else begin fnotroot ← TRUE                           (194)
            end                                                      (195)
        end                                                          (196)
        if cnt2 > 1 then return FALSE                                (197)
        if fnotroot =FALSE & froot =TRUE then return FALSE           (198)
    end                                                              (199)
    return TRUE                                                      (200)
end
```

Figure A.2: Procedure for testing rootedness

(c) Two or more roots are not gluing pivots.
In this case, they become roots in a glued graph.

Only (a-2) and (b-1) produce an LDG with only one root, and they correspond to (1) and (2) in the rootedness conditions, respectively.

Therefore, if given RLDAGs satisfy the rootedness conditions, a glued graph of them becomes a rooted LDG.

□

Procedures for testing for rootedness conditions 1 and 2 are shown in Figure A.2. These procedures also use the function *getGluingNodeSet* in order to collect gluing node sets. The procedure *testRootedness*1 tests whether condition 1 holds or not. It returns false if there is a root whose gluing node set contains only the root itself (line 170), which corresponds to the first part of condition 1. Further, *cnt*1 represents the number of gluing node sets whose elements are all roots; if it is greater than 1, this procedure returns false (line 181), because it violates the latter part of condition 1. Lines 173-178 check whether such a node set exists. The procedure *testRootness*2 tests whether the condition 2 holds or not. *cnt*2 is the number of roots that are not gluing pivots; if it is greater than 1 this procedure returns false (line 197), because it violates the first part of condition 2. *froot* represents whether there is a root node in the current gluing node set; *fnotroot*, on the other hand, represents whether there is a node that is not a root in the current gluing node set. If *froot* is true and *fnotroot* is false, the procedure returns false (line 198), because it violates the second part of condition 2.

Let us consider the complexity of these procedures. The complexity of the function *getGluingNodeSet* is $O(k^2n^2)$. Since the maximal value of *lnum* is kn and the maximal number of nodes of $N_g[]$ is kn, the outer loop iterates at most kn times and the inner loop iterates at most kn times for both procedures. Therefore, the complexity of the main loop of both procedures is $O(k^2n^2)$. Therefore, the total complexity of both procedures is $O(k^2n^2)$.

Hence, if the given RLDAGs satisfy the above conditions for acyclicity and rootedness, a glued graph of them becomes an RLDAG.

Bibliography

[Abeillé et al. 90] Abeillé, A., Schabes, Y., and Joshi, A. K., "Using Lexicalized Tags for Machine Translation," *Proc. of COLING 90*, pp. 1-6, 1990.

[Ait-Kaci 86] Ait-Kaci, H., "An Algebraic Semantics Approach to the Effective Resolution of Type Equations," *Theoretical Computer Science 45*, pp. 293-351, 1986.

[Boitet and Nedobejkine 81] Boitet, C. and Nedobejkine, N., "Recent Development in Russian-French Machine Translation at Grenoble," *Linguistics 19*, pp. 199-271, 1981

[Ehrig 79] Ehrig, H., "Introduction to the Algebraic Theory of Graph Grammars," *Proc. of International Workshop on Graph Grammars, LNCS 73*, pp. 1-69, 1979.

[Furuse and Iida 92] Furuse, O. and Iida, H., "Cooperation between Transfer and Analysis in Example-Based Framework," *Proc. of COLING 92*, Vol. 2, pp. 645-651, 1992.

[Gazdar et al. 85] Gazdar, G., Klein. E., Pullum, G. K., and Sag, I. A., "Generalized Phrase Structure Grammar," Harvard University Press, 1985.

[Hutchins 86] Hutchins, W. J. (ed.) : MACHINE TRANSLATION: Past, Present, Future, pp. 239-248, Ellis Horwood Limited, 1986.

[Kaji et al. 92] Kaji, H., Kida, Y., and Morimoto, Y., "Learning Translation Templates from Bilingual Text," *Proc. of COLING 92*, Vol. 2, pp. 672-678, 1992.

[Kaplan and Bresnan 82] Kaplan, R. and Bresnan, J., "Lexical Functional Grammar: a Formal System for Grammatical Representation," (Bresnan, J. ed:) *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, Massachusetts, pp. 173-281, 1982.

[Kay 85] Kay, M., "Parsing in Functional Unification Grammar," *Psychological, Computational and Theoretical Perspectives*, pp. 251-278, 1985.

[Kitano and Higuchi 91] Kitano, H. and Higuchi, T., "Massively Parallel Memory-Based Parsing," *Proc. of IJCAI 91*, pp. 918-924, 1991.

[Knight 89] Knight, K., "Unification: A Multidisciplinary Survey," *ACM Computing Surveys*, Vol. 21, No. 1, pp. 93-124, March 1989.

[Kolodner and Riesbeck 89] Kolodner, J. and Riesbeck, C., "Case-Based Reasoning," tutorial textbook of 11th IJCAI, 1989.

[Kudo and Nomura 86] Kudo, I. and Nomura, H., "Lexical-Functional Transfer: A Transfer Framework in a Machine Translation System Based on LFG," *Proc. of COLING 86*, pp. 112-114,1986.

[Maruyama and Watanabe 92] Maruyama, H. and Watanabe, H., "Tree Cover Search Algorithm for Example-Based Translation," *Proc. of 4th Int. Conf. on Theoretical and Methodological Issues in Machine Translation*, pp. 173-184, 1992.

[Matsumoto et al. 93] Matsumoto, Y., Ishimoto, H., and Utsuro, T., "Structural Matching of Parallel Text," *Proc. of 31st Annual Meeting of ACL*, pp. 23-30, 1993.

[Melby 86] Melby, A. K., "Lexical Transfer: A Missing Element in Linguistic Theories," *Proc. of COLING 86*, pp. 104-106, 1986.

[MTJ 89a] Nirenburg, S. (ed.) : Machine Translation (Special issue on knowledge-based machine translation, Part I), Vol. 4, No. 1, Kluwer Academic Press, 1989.

[MTJ 89b] Nirenburg, S. (ed.) : Machine Translation (Special issue on knowledge-based machine translation, Part II), Vol. 4, No. 2, Kluwer Academic Press, 1989.

[MTJ 91a] Nirenburg, S. (ed.) : Machine Translation (Special issue on EUROTRA I), Vol. 6, No. 2, Kluwer Academic Press, 1991.

[MTJ 91b] Nirenburg, S. (ed.) : Machine Translation (Special issue on EUROTRA II), Vol. 6, No. 3, Kluwer Academic Press, 1991.

[Nagao 84] Nagao, M., "A Framework of a Mechanical Translation between Japanese and English by Analogy Principle," Elithorn, A. and Banerji, R. (eds.) : *Artificial and Human Intelligence*, North-Holland, pp. 173-180, 1984.

[Nagao and Tsujii 86] Nagao, M. and Tsujii, J., "The Transfer Phase of the Mu Machine Translation System," *Proc. of COLING 86*, pp. 97-103, 1986.

[Nagao 92] Nagao, M., "Some Rationales and Methodologies for Example-Based Approach," *Proc. of 2nd FGNLP workshop*, pp. 82-94, 1984.

[Nitta 86] Nitta, Y., "Idiosyncratic Gap: A Tough Problem to Structure-bound Machine Translation," *Proc. of COLING 86*, pp. 107-111, 1986.

[NLRI 64] National Language Research Institute: Word List by Semantic Principles (in Japanese), Syuei Syuppan, 1964.

[Nomiyama 92] Nomiyama, H., "Machine Translation by Case Generalization," *Proc. of COLING 92*, Vol. 2, pp. 714-720, 1992.

[Perlmutter 83] Perlmutter, D. M. (ed.) : Studies in Relational Grammar 1, Chicago: University of Chicago Press, 1983.

[Perlmutter and Rosen 84] Perlmutter, D. M., and Rosen, C. G. (eds.) : Studies in Relational Grammar 2, Chicago: University of Chicago Press, 1984.

[Salder 89] Sadler, V., "The Bilingual Knowledge Bank," BSO Research, March 1989.

[Sato and Nagao 90] Sato, S. and Nagao, M., "Toward Memory-Based Translation," *COLING 90*, Vol. 3, pp. 247-252, 1990.

[Schenk 86] Schenk, A., "Idioms in the ROSETTA Machine Translation System," *Proc. of COLING 86*, pp. 319-324, 1986.

[Sowa 84] Sowa, J. F., "Conceptual Structures: Information Processing in Mind and Machine," Addison-Wesley Publishing, 1984.

[Sumita et al. 90] Sumita, E., Iida, H., and Kohyama, H., "Translating with Examples: A New Approach to Machine Translation," *Proc. of Info Japan 90*, 1990.

[Sumita and Iida 91] Sumita, E., and Iida, H., "Experiments and Prospects of Example-Based Machine Translation," *Proc. of 29th ACL*, pp. 185-192, 1991.

[Tomita and Carbonell 86] Tomita, M. and Carbonell, J. G., "Another Stride Towards Knowledge-Based Machine Translation," *Proc. of COLING 86*, pp. 633-638, 1986

[Tsujii and Fujita 90] Tsujii, J. and Fujita, K., "Lexical Transfer Based on Bilingual Signs: Towards Interaction during Transfer," *Proc. of Seoul Int. Conf. on NLP*, 1990.

[Uramoto 91] Uramoto, N, "Lexical and Structural Disambiguation Using an Example-Base," *Proc. of the 2nd Japan-Australia Joint Symposium on NLP*, pp. 150-160, 1991.

[Utsuro et al. 92] Utsuro, T., Matsumoto, Y., and Nagao, M., "Lexical Knowledge Acquisition from Bilingual Corpora," Proc. of COLING 92, Vol. 2, pp. 581-587, 1992.

[Utsuro et al. 94] Utsuro, T., Ikeda, H., Yamane, M., Matsumoto, Y., and Nagao, M., "Bilingual Text Matching Using Bilingual Dictionary and Statistics," Proc. of COLING 94, Vol. 2, pp. 1076-1082, 1994.

[Whitelock 92] Whitelock, P., "Shake-and-Bake Translation," *Proc. of COLING 92*, Vol. 2, pp. 784-791, 1992

List of Publications

List of Major Publications

[1] Watanabe, H., "A Model of a Transfer Process Using Combinations of Translation Rules," *Proc. of Pacific Rim International Conference on Artificial Intelligence '90*, pp. 215-220, 1990.

[2] Maruyama, H. and Watanabe, H., "Tree Cover Search Algorithm for Example-Based Translation," *Proc. of the 4th International Conference on Theoretical and Methodological Issues in Machine Translation*, pp. 173-184, 1992.

[3] Watanabe, H., "A Similarity-Driven Transfer System," *Proc. of the 14th International Conference of Computational Linguistics*, Vol. 2, pp. 770-776, 1992.

[4] Watanabe, H., "A Method for Extracting Translation Patterns from Translation Examples," *Proc. of 5th International Conference on Theoretical and Methodological Issues in Machine Translation*, pp. 292-301, 1993.

[5] Watanabe, H. and Maruyama, H., "A Transfer System Using Example-Based Approach," *IEICE Transactions on Information and Systems*, Vol. E77-D, No. 2, pp. 247-257, 1994.

[6] Watanabe, H., "A Method for Distinguishing Exceptional and General Examples in Example-Based Transfer Systems," *Proc. of the 15th International Conference of Computational Linguistics*, Vol. 1, pp. 39-44, 1994.

[7] Watanabe, H., "A System for Finding Translation Patterns by Comparing an MT Result and Its Correction," *Journal of Natural Language Processing*, Vol. 1, No. 1, pp. 59-75, 1994.

[8] Watanabe, H., "A Formal Model of Transfer Using Rule Combination," *Machine Translation Journal* Vol. 10, No. 4, pp. 269 291, 1995.

List of Other Publications

[1] Hideo Watanabe, Jun-ichi Tsujii, and Makoto Nagao, "A Method for Analyzing Text Structure by Using Surface Clues in Sentences" (in Japanese), Proc. of 32nd Convention of IPSJ, Vol. 2, pp. 1633 1634, 1986.

[2] Hideo Watanabe and Hiroshi Maruyama, "A Natural Language Interface System Which Can Respond To Many Applications" (in Japanese), IPSJ-WGNL, 65-3, 1988.

[3] Hideo Watanabe and Hiroshi Maruyama, "A System for Analyzing Japanese Sentences Interactively Based on Constraint Dependency Grammar" (in Japanese), IPSJ-WGNL, 69-6, 1988

[4] Hideo Watanabe and Hiroshi Maruyama, "An Interactive Japanese Analysis Environment: JAWB" (in Japanese), Proc. of 38th Convention of IPSJ, Vol. 1, pp. 396 397, 1989.

[5] Hideo Watanabe, "A Transducing Method for Similarity-driven Translation System" (in Japanese), Proc. of 42nd Convention of IPSJ, Vol. 3, pp. 19-20, 1991.

[6] Hideo Watanabe, "A Transfer System Combining Similar Translation Patterns" (in Japanese), Proc. of 8th Annual Convention of JSSST, pp. 185-188, 1991.

[7] Hideo Watanabe, "A Transfer System Using Translation Examples Including Syntactic Information" (in Japanese), Proc. of 43rd Convention of IPSJ, Vol. 3, 193 194, 1991

[8] Hideo Watanabe and Naohiko Uramoto, "Issues on Example-Based Machine Translation" (in Japanese), Proc. of 45th Convention of IPSJ, Vol. 3, pp. 105 106, 1992.

[9] Taijiro Tsutsumi, Hideo Watanabe, Hiroshi Maruyama, Naohiko Uramoto, Masayuki Morohashi, Koichi Takeda, and Tetsuya Nasukawa, "Example-Based

Approach to Machine Translation," Proc. of Premières Journées Franco-Japonaises sur la Traduction Assistée par Ordinateur, pp. 15 16, 1993

[10] Hideo Watanabe, "A Method for Extracting Translation Patterns Automatically from Translation Examples for Example-Based Translation" (in Japanese), Proc. of 47th Convention of IPSJ, Vol. 3, pp. 203-204, 1993.

[11] Hideo Watanabe, "An Assisting System for Creating Translation Patterns in Japanese-to-English MT System JETS" (in Japanese), Proc. of 49th Convention of IPSJ, Vol. 3, pp. 195 196, 1994.

[12] Hideo Watanabe, "A Method for Abstracting Newspaper Articles" (in Japanese), Proc. of 1st Annual Meeting of ANLP, pp. 293-296, 1995.

Abbreviations

ACL Association for Computational Linguistics

ANLP Association for Natural Language Processing

COLING International Conference on Computational Linguistics

IEICE Institute of Electronics, Information and Communication Engineers

IJCAI International Joint Conference on Artificial Intelligence

IPSJ Information Processing Society of Japan

WGNL Working Group on Natural Language Processing

JSAI Japan Society for Artificial Intelligence

JSSST Japan Society for Software Science and Technology

MTJ Machine Translation Journal

PRICAI Pacific Rim International Conference on AI

TMI International Conference on Theoretical and Methodological Issues in Machine Translation